计算机科学丛书

华章教育

McGraw Hill Education

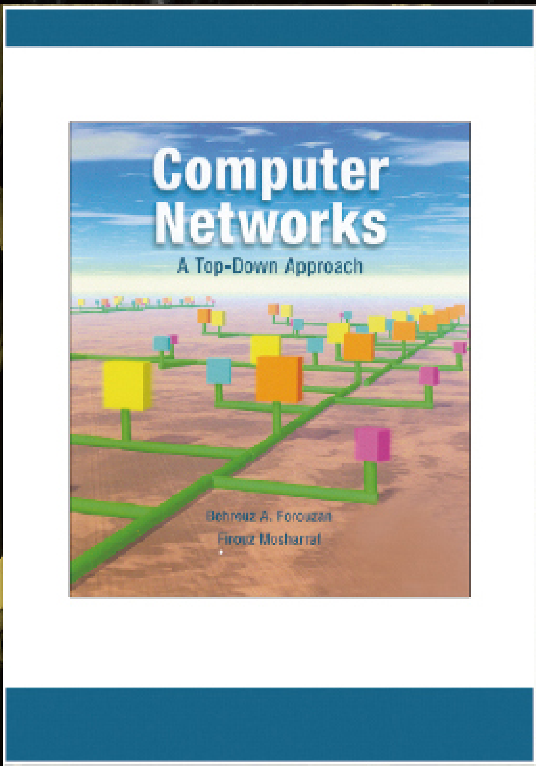
计算机网络教程

自顶向下方法

（美）Behrouz A. Forouzan　Firouz Mosharraf 著
张建忠 靳星 林安华 周立斌　译

Computer Networks

A Top-Down Approach



机械工业出版社
China Machine Press

VISIT…

LANZAROTE Caliente.com

计算机科学丛书

计算机网络教程
自顶向下方法

Computer Networks: A Top-Down Approach

（美）Behrouz A. Forouzan
Firouz Mosharraf 著

张建忠　靳　星　林安华　周立斌　译

HZ Books
华章图书

机械工业出版社
China Machine Press

本书作者 Forouzan 是计算机教育领域的知名专家，他在这本经典著作中，利用 Internet 协议分层和 TCP/IP 协议簇，采用自顶向下的方法，首先说明应用层协议是怎样交换信息的，再解释消息是怎样分解成比特和信号并通过 Internet 传输的，以通俗易懂的方式阐述了计算机网络原理，帮助学生从总体上理解网络的基础知识，特别是 Internet 的协议。

本书图文并茂，实例丰富，并配有大量的习题集（包括测试题、练习题、思考题）以及模拟实验和编程作业，适合作为本科生、研究生的计算机网络教材，同时也适合计算机网络研究和专业人员阅读。

Behrouz A. Forouzan, Firouz Mosharraf: *Computer Networks: A Top-Down Approach* (ISBN 978-0-07352326-2).

Copyright © 2011 by The McGraw-Hill Companies, Inc.

All Rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including without limitation photocopying, recording, taping, or any database, information or retrieval system, without the prior written permission of the publisher.

This authorized Chinese translation edition is jointly published by McGraw-Hill Education (Asia) and China Machine Press. This edition is authorized for sale in the People's Republic of China only, excluding Hong Kong, Macao SAR and Taiwan.

Copyright © 2013 by McGraw-Hill Education (Asia), a division of McGraw-Hill Asian Holdings (Singapore) Pte.Ltd. And China Machine Press.

版权所有。未经出版人事先书面许可，对本出版物的任何部分不得以任何方式或途径复制或传播，包括但不限于复印、录制、录音，或通过任何数据库、信息或可检索的系统。

本授权中文简体字翻译版由麦格劳-希尔（亚洲）教育出版公司和机械工业出版社合作出版。此版本经授权仅限在中华人民共和国境内（不包括香港特别行政区、澳门特别行政区和台湾）销售。

版权© 2013 由麦格劳-希尔（亚洲）教育出版公司与机械工业出版社所有。

本书封面贴有 McGraw-Hill 公司防伪标签，无标签者不得销售。

封底无防伪标均为盗版
版权所有，侵权必究
本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2011-5060

图书在版编目（CIP）数据

计算机网络教程：自顶向下方法 / （美）佛罗赞（Forouzan, B. A.），（美）莫沙拉夫（Mosharraf, F.）著；张建忠等译. —北京：机械工业出版社，2012.10
（计算机科学丛书）
书名原文：Computer Networks：A Top-Down Approach

ISBN 978-7-111-40088-2

Ⅰ. 计…　Ⅱ. ①佛…　②莫…　③张…　Ⅲ. 计算机网络-教材　Ⅳ. TP393

中国版本图书馆 CIP 数据核字（2012）第 245205 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）
责任编辑：刘立卿
印刷
2013 年 1 月第 1 版第 1 次印刷
185mm×260mm • 39 印张
标准书号：ISBN 978-7-111-40088-2
定价：99.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

出版者的话

Computer Networks: A Top-Down Approach

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage 等世界著名出版公司建立了良好的合作关系， 从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037

HZ BOOKS 华章教育

华章科技图书出版中心

译　者　序

Computer Networks: A Top-Down Approach

近年来，计算机网络技术已经应用到各个领域。人们的学习、工作和生活已经渐渐离不开计算机网络，因此需要了解和掌握计算机网络技术的群体在不断扩大。不论是计算机及相关专业的学生，还是从事相关工作的工程技术人员，都需要较深入地了解计算机网络的基本原理和应用形式。本书作为一本深入浅出且广泛适用的计算机网络教材，能够帮助读者理解和掌握复杂的网络知识。

Behrouz A. Forouzan 是计算机教育领域的知名作家，他出版了《Data Communications and Networking》、《TCP/IP Protocol Suite》、《Cryptography & Network Security》、《Foundations of Computer Science》等多部畅销教材，涉猎范围包括计算机科学领域的多个方面。他与时俱进，作品紧跟计算机技术和计算机教育的步伐。《Computer Network：A Top-Down Approach》是 Forouzan 按照目前计算机网络教学比较流行的自顶向下方法编写的一部重要教材，与《Data Communications and Networking》相比，本书在内容上增加了一些目前计算机网络发展的新技术，编排上采用自顶向下的结构，这样既适合学生学习，又适合教师教授。

暑假前夕，出版社的同志和我们联系，希望我们能够翻译这本书。由于要求的翻译周期很短，所以一开始我们非常犹豫。但是，作为教师，我们深知优秀教材在“教”与“学”中的作用。在经过认真思考之后，我们最终欣然接受了这项具有挑战性的任务。接下来的四个月，翻译工作花费了我们大量的时间和精力。

本书共分为 11 章，第 1 章和第 10 章由张建忠翻译，第 2 章、第 3 章和第 4 章由靳星翻译，第 5 章、第 6 章和第 11 章由林安华翻译，第 7 章、第 8 章和第 9 章由周立斌翻译。张建忠负责最后统稿。受译者水平和翻译周期所限，译稿中可能存在不当或错误之处，敬请广大读者批评指正。

译者

2012 年 8 月于南开园

华章图书

当今社会，网络与互联网技术迅猛发展，其佐证就是每年各种新型社交网络应用的不断涌现。人们每天使用 Internet 的频率越来越高，他们利用 Internet 进行科学研究、网络购物、机票预订、查看新闻与天气状况……

在这个面向 Internet 的社会中，需要培养和训练专业技术人员对 Internet、部分 Internet 或者连接到 Internet 的内部网络进行运营和管理。本书的目标是帮助学生总体上理解网络的基本知识，特别是 Internet 使用的协议。

本书特色

本书的主要目标是讲授网络原理，为了讲授这些原理，本书采用了以下方法。

协议分层

本书利用 Internet 协议分层和 TCP/IP 协议簇讲授网络原理。虽然有些网络理论在某些层次上可能有些重复，但每层都会有其特别强调的方面。这些理论在不同层次重复出现，使我们能够更好地理解相互之间的关系，因此利用协议分层方法讲授网络理论是有益的。例如，虽然**寻址**（addressing）在 TCP/IP 的 4 个层次中都会遇到，但是各层使用了不同的地址格式以实现各自不同的目标。另外，每层中的寻址范围也有不同。另一个例子是**成帧与分组**（framing and packetizing），这些内容在几层中也会重复出现，但是各层涉及的理论不同。

自顶向下方法

尽管本书的作者之一曾经编写过几本与网络和 Internet 相关的书籍（《Data Communication and Networking》、《TCP/IP Protocol Suite》、《Cryptography and Network Security》、《Local Area Networks》），但是本书讲授网络的方法不同。它采用自顶向下的方法。

虽然 TCP/IP 协议每一层建立在它下层提供的服务之上，但是学习每一层的知识有两种方法——自底向上或自顶向下。自底向上方法中，我们在学习应用层怎样利用比特传送消息之前，学习比特和信号怎样在物理层移动。自顶向下方法中，我们在学习消息怎样被分解成比特和信号、怎样实际地通过 Internet 传送之前，学习应用层协议怎样交换信息。在本书中，我们采用了自顶向下方法。

面向的读者

本书面向的读者为学术和专业技术人员。感兴趣的专业技术人员也可用本书作为自学指导书。作为教材，它可以用于一个学期或半个学期的课程，适用于大学本科最后一学年或研究生第一年的学习。虽然章节末尾的思考题需要一些概率知识，但是教材的学习只需要大学一年级讲授的基本数学知识。

本书的组织

本书包括 11 章和 5 个附录。

- 第 1 章　概论
- 第 2 章　应用层
- 第 3 章　传输层
- 第 4 章　网络层
- 第 5 章　数据链路层：有线网络
- 第 6 章　无线网络和移动 IP
- 第 7 章　物理层和传输介质
- 第 8 章　多媒体和服务质量
- 第 9 章　网络管理
- 第 10 章　网络安全
- 第 11 章　Java Socket 编程
- 附录　附录 A 到附录 E

写作方法

本书采用的几种写作方法使学生能够很容易地理解计算机网络的基础知识，特别是 Internet 的相关知识。

形象直观

本书采用图文并茂的方式描述技术性很强的问题，而没有采用复杂的数学公式。670 多幅插图与文字讲解，使内容更加直观易懂。在解释网络概念时，插图的作用尤其重要。对于很多学生来说，通过插图理解这些概念比通过文字更容易。

举例和应用

在合适的位置我们加入了一些例子，用于说明书中介绍的相关概念。同时，我们也在每章中添加了一些现实中的应用，用于激励学生学习。

章末资料

每章后包含的相关资料如下：

小结　每章末尾都包含覆盖本章内容的小结。小结将本章的重点内容关联起来，一目了然。

推荐读物　这一部分列出了与本章内容相关的主要参考文献。利用这个参考文献列表，可以在书末尾的“参考文献”部分快速找到相应文献。

习题集

每章都设计有习题集，用于巩固重要的概念，同时鼓励学生应用这些概念。习题集包括 3 部分：测试题、练习题和思考题。

测试题　测试题放置在本书的网站中，用于快速检查概念的掌握情况。学生可以通过完成这些测试题查看自己对内容的理解程度，网站可立即给出测试结果。

练习题　这部分包含与本书讨论内容相关的一些简单问题。题号为奇数的练习题答案放

置于本书的网站中，学生可以查阅。

思考题　这部分内容包括一些较难的问题，需要较为深入地理解本章的内容才能解答。我们强烈推荐学生尝试解决所有这些问题。题号为奇数的思考题答案也放置于本书的网站中，学生可以查阅。

模拟实验

如果能够动手对分组流和分组内容进行分析，那么就能更好地理解这些网络概念。多数章节包含了一部分用于帮助学生进行实验的内容。这一部分内容分为两部分。

Applets　Java 小程序（Applets）放置在网站上，是由作者设计的交互式实验。这些小程序一部分用于更好地理解一些问题的解决方案，另一部分用于帮助读者在动手操作中更好地理解网络概念。

实验作业　一些章节包含了使用 Wireshark 模拟软件的实验作业。下载和使用 Wireshark 软件的方法在第 1 章中给出。另外一些章节给出的实验作业用于练习发送和接收分组，同时分析这些分组的内容。

编程作业

一部分章节包含有编程作业。编写一个有关进程或过程的程序能够澄清很多细节，并且能够帮助学生更好地理解隐藏在进程之后的概念。虽然学生可以使用自己熟悉的任意一种计算机语言编写和测试程序，但是本书网站中给出的答案是使用 Java 语言编写的，这些答案供教师使用。

附录

附录的目的是提供快速的资料参考和内容回顾，这些资料和内容可用于理解本书讨论的概念。

术语表和索引

为了更快地检索词汇和缩略语，本书给出了术语表和索引，但因篇幅所限，这些材料不包含在中文版书中，读者可到华章网站 http://www.hzbook.com 查阅。

教辅资源

本书包含的完整教辅资源可以从本书的网站 http://www.mhhe.com/forouzan 中下载[⊖]。这些资源包括以下内容。

幻灯片

网站给出了一组华美且栩栩如生的 PowerPoint 幻灯片，用于教学使用。

习题集答案

本书网站提供所有练习题和思考题的答案，供教师教学使用。

编程作业答案

本书网站也提供编程作业答案。其中第 2 章的程序采用 C 语言编写，其他章节的程序采用 Java 语言编写。

⊖ 采用该书作教材的教师可向 McGraw-Hill 公司北京代表处联系索取教学课件资料，传真：+8610-62790292；电子邮件：instructorChina@mcgraw-hill.com。

如何使用本书

本书章节的组织提供了很大的灵活性，我们建议如下：

- 第 1 章讨论的大部分内容是理解本书其他章节内容的基础。1.1 节和 1.2 节内容对理解网络分层非常关键，而网络分层是本书内容组织的基础。1.3 节和 1.4 节可以跳过或者安排为自学内容。
- 第 2 章至第 6 章基于 TCP/IP 协议簇的顶部 4 层，我们建议按照次序讲授，以保持本书自顶向下的方法。可是，有些部分可以跳过而不会失去连续性，例如第 2 章的客户-服务器 Socket 接口、第 4 章的下一代 IP、第 5 章的其他有线网络。
- 为了使 TCP/IP 协议的讨论更加完整，本书添加了第 7 章物理层。如果教师感觉学生已经熟悉或者已经在相关课程中学习过这些内容，那么这些内容可以跳过。
- 在前 6 章讨论完后，第 8 章、第 9 章和第 10 章可以按任意次序讲授。教师可以全部或部分地讲授这些章节的内容，甚至可以跳过这些内容。
- 第 11 章为 Java 网络编程。该章有两个目的：首先，它给出客户-服务器编程思想，使学生更好地理解 Internet 的整体目标。其次，它可以为网络方面的高级课程做准备。第 2 章中关于 C 语言的内容与本章有一小部分重复，教师既可以使用第 2 章 C 语言部分的内容讲授网络编程基础，也可以使用第 11 章 Java 语言的内容进行讲授。

本书网站

本书网站 http://www.mhhe.com/forouzan 包含以下内容。

测试题

测试题放置于本书网站中，测试结果可以发送给讲授该课程的教师。

学生答案

奇数题号练习题和思考题的答案放置在本书网站中，用于帮助学生检查他们的学习状况。

Applets

学生可以使用为每章设计的小程序，观察实际的网络协议及其问题。

教师答案

所有练习题和思考题的答案放置在本书网站中，供讲授本课程的教师使用。

编程作业

编程作业的代码放置在本书网站中，供讲授本课程的教师使用。

PowerPoint 幻灯片

华美且栩栩如生的幻灯片放置在本书网站中，供讲授本课程的教师使用。教师可以修改这些幻灯片以适应课程的需要。

致谢

显然，编写如此篇幅的书籍需要很多人的帮助。我们非常感谢同行评审专家在本书编写过程中做出的贡献。这些评审专家为：

HZ BOOKS 华章图书

Zongming Fei	肯塔基大学（University of Kentucky）
Randy J. Fortier	温莎大学（University of Windsor）
Seyed H. Hosseini	威斯康星大学米尔沃基分校（University of Wisconsin,Milwaukee）
George Kesidis	宾夕法尼亚州立大学（Pennsylvania State University）
Amin Vahdat	加利福尼亚大学圣地亚哥分校（University of California, San Deigo）
Yannis Viniotis	北卡罗来纳州立大学（North Carolina State University）
Bin Wang	莱特州立大学（Wright State University）
Vincent Wong	英属哥伦比亚大学（University of British Columbia）
Zhi-Li Zhang	明尼苏达大学（University of Minnesota）
Wenbing Zhao	克利夫兰州立大学（Cleveland State University）

特别感谢 McGraw-Hill 出版公司的人员。出版人 Raghu Srinivasan 证明了出版专家可以将不可能的事情变成可能。无论何时，只要有需要，策划编辑 Melinda Bilecki 都会给予帮助。在整个出版过程中，项目经理 Jane Mohr 一直以极大的热情指导我们。我们还要感谢项目经理 Dheeraj Chahal、封面设计人 Brenda A. Rolwes 和文字编辑 Kathryn DiBernardo。

Forouzan 和 Mosharraf

加利福尼亚，洛杉矶

HZ BOOKS
华章图书

目　录

Computer Networks: A Top-Down Approach

出版者的话

译者序

前言

第 1 章　概论 …… 1

1.1　Internet 概览 …… 1

1.1.1　网络 …… 1

1.1.2　交换 …… 3

1.1.3　Internet …… 5

1.1.4　访问 Internet …… 6

1.1.5　硬件和软件 …… 6

1.2　协议分层 …… 6

1.2.1　场景 …… 7

1.2.2　TCP/IP 协议簇 …… 8

1.2.3　OSI 模型 …… 15

1.3　Internet 发展史 …… 16

1.3.1　早期历史 …… 16

1.3.2　Internet 的诞生 …… 16

1.3.3　今天的 Internet …… 17

1.4　标准和管理 …… 18

1.4.1　Internet 标准 …… 18

1.4.2　Internet 管理 …… 19

1.5　章末资料 …… 20

推荐读物 …… 20

小结 …… 20

1.6　习题集 …… 21

测试题 …… 21

练习题 …… 21

思考题 …… 22

1.7　模拟实验 …… 23

Applets …… 23

实验作业 …… 23

第 2 章　应用层 …… 24

2.1　介绍 …… 24

2.1.1　提供服务 …… 24

2.1.2　应用层模式 …… 26

2.2　客户–服务器模式 …… 28

2.2.1　应用程序接口 …… 28

2.2.2　使用传输层的服务 …… 30

2.3　标准客户–服务器应用 …… 31

2.3.1　万维网和 HTTP …… 32

2.3.2　FTP …… 42

2.3.3　电子邮件 …… 45

2.3.4　TELNET …… 55

2.3.5　安全 Shell …… 57

2.3.6　域名系统 …… 58

2.4　对等模式 …… 66

2.4.1　P2P 网络 …… 66

2.4.2　分布式散列表 …… 68

2.4.3　Chord …… 70

2.4.4　Pastry …… 75

2.4.5　Kademlia …… 79

2.4.6　一种流行的 P2P 网络：BitTorrent …… 81

2.5　套接字接口编程 …… 83

C 的套接字接口 …… 83

2.6　章末资料 …… 94

推荐读物 …… 94

小结 …… 95

2.7　习题集 …… 95

测试题 …… 95

练习题 …… 95

思考题 …… 97

2.8　模拟实验 …… 99

Applets …… 99

实验作业 …… 99

2.9　编程作业 …… 99

第 3 章　传输层 …… 100

3.1　介绍 …… 100

传输层服务 …… 100

3.2　传输层协议 …… 110

3.2.1 简单协议 ………… 111
3.2.2 停止-等待协议 ………… 111
3.2.3 回退 N 帧协议 ………… 115
3.2.4 选择性重复协议 ………… 120
3.2.5 双向协议：捎带 ………… 123
3.2.6 因特网传输层协议 ………… 124
3.3 用户数据报协议 ………… 125
3.3.1 用户数据报 ………… 126
3.3.2 UDP 服务 ………… 126
3.3.3 UDP 应用 ………… 128
3.4 传输控制协议 ………… 129
3.4.1 TCP 服务 ………… 130
3.4.2 TCP 的特点 ………… 131
3.4.3 段 ………… 133
3.4.4 TCP 连接 ………… 134
3.4.5 状态转换图 ………… 139
3.4.6 TCP 中的窗口 ………… 141
3.4.7 流量控制 ………… 143
3.4.8 差错控制 ………… 147
3.4.9 TCP 拥塞控制 ………… 152
3.4.10 TCP 计时器 ………… 159
3.4.11 选项 ………… 162
3.5 章末资料 ………… 162
推荐读物 ………… 162
小结 ………… 162
3.6 习题集 ………… 163
测试题 ………… 163
练习题 ………… 163
思考题 ………… 165
3.7 模拟实验 ………… 169
Applets ………… 169
实验作业 ………… 169
3.8 编程作业 ………… 169

第 4 章 网络层 ………… 170
4.1 介绍 ………… 170
4.1.1 网络层服务 ………… 170
4.1.2 分组交换 ………… 173
4.1.3 网络层性能 ………… 177
4.1.4 网络层拥塞 ………… 179
4.1.5 路由器的结构 ………… 182
4.2 网络层协议 ………… 183
4.2.1 IPv4 数据报格式 ………… 184
4.2.2 IPv4 地址 ………… 189
4.2.3 IP 分组的转发 ………… 202
4.2.4 ICMPv4 ………… 208
4.3 单播路由选择 ………… 211
4.3.1 一般思想 ………… 212
4.3.2 路由选择算法 ………… 213
4.3.3 单播路由选择协议 ………… 222
4.4 多播路由选择 ………… 237
4.4.1 介绍 ………… 237
4.4.2 多播基础 ………… 239
4.4.3 域内路由选择协议 ………… 243
4.4.4 域间路由选择协议 ………… 248
4.5 下一代 IP ………… 248
4.5.1 分组格式 ………… 249
4.5.2 IPv6 寻址 ………… 251
4.5.3 从 IPv4 到 IPv6 的过渡 ………… 254
4.5.4 ICMPv6 ………… 255
4.6 章末资料 ………… 257
推荐读物 ………… 257
小结 ………… 257
4.7 习题集 ………… 258
测试题 ………… 258
练习题 ………… 258
思考题 ………… 260
4.8 模拟实验 ………… 264
Applets ………… 264
实验作业 ………… 264
4.9 编程作业 ………… 264

第 5 章 数据链路层：有线网络 ………… 265
5.1 介绍 ………… 265
5.1.1 结点和链路 ………… 265
5.1.2 两类链路 ………… 267
5.1.3 两个子层 ………… 267
5.2 数据链路控制 ………… 267
5.2.1 成帧 ………… 267
5.2.2 流量控制和差错控制 ………… 269
5.2.3 差错检测和纠错 ………… 270
5.2.4 两种 DLC 协议 ………… 280
5.3 多路访问协议 ………… 285
5.3.1 随机访问 ………… 285

HZ BOOKS 华章图书

5.3.2 受控访问 ……294
5.3.3 通道化 ……296
5.4 链路层寻址 ……296
5.5 有线局域网：以太网协议 ……303
5.5.1 IEEE 项目 802 ……304
5.5.2 标准以太网 ……304
5.5.3 快速以太网（100 Mbps）……309
5.5.4 千兆以太网 ……310
5.5.5 10 千兆以太网 ……310
5.5.6 虚拟局域网 ……310
5.6 其他有线网络 ……313
5.6.1 点对点网络 ……313
5.6.2 SONET ……317
5.6.3 交换网络：ATM ……322
5.7 连接设备 ……325
5.7.1 中继器或集线器 ……325
5.7.2 链路层交换机 ……326
5.7.3 路由器 ……327
5.8 章末资料 ……328
推荐读物 ……328
小结 ……328
5.9 习题集 ……329
测试题 ……329
练习题 ……329
思考题 ……331
5.10 模拟实验 ……335
Applets ……335
实验作业 ……335
5.11 编程作业 ……335

第 6 章 无线网络和移动 IP ……336
6.1 无线局域网 ……336
6.1.1 介绍 ……336
6.1.2 IEEE 802.11 项目 ……339
6.1.3 蓝牙 ……347
6.1.4 WiMAX ……352
6.2 其他无线网络 ……353
6.2.1 通道化 ……353
6.2.2 蜂窝电话 ……358
6.2.3 卫星网络 ……366
6.3 移动 IP ……369
6.3.1 寻址 ……369
6.3.2 代理 ……370
6.3.3 三个阶段 ……371
6.3.4 移动 IP 的低效 ……374
6.4 章末资料 ……375
推荐读物 ……375
小结 ……376
6.5 习题集 ……376
测试题 ……376
练习题 ……376
思考题 ……377
6.6 模拟实验 ……380
Applets ……380
实验作业 ……380
6.7 编程作业 ……380

第 7 章 物理层与传输介质 ……381
7.1 数据和信号 ……381
7.1.1 模拟数据与数字数据 ……381
7.1.2 传输减损 ……387
7.1.3 数据速率限制 ……389
7.1.4 性能 ……390
7.2 数字传输 ……392
7.2.1 数字到数字转换 ……392
7.2.2 模拟到数字转换 ……397
7.3 模拟传输 ……400
7.3.1 数字到模拟转换 ……400
7.3.2 模拟到模拟转换 ……404
7.4 带宽利用 ……405
7.4.1 多路复用 ……405
7.4.2 扩频 ……410
7.5 传输介质 ……412
7.5.1 有向介质 ……412
7.5.2 无向介质：无线 ……416
7.6 章末资料 ……417
推荐读物 ……417
小结 ……417
7.7 习题集 ……418
测试题 ……418
练习题 ……418
思考题 ……419

第 8 章 多媒体和服务质量 ……423
8.1 压缩 ……423

8.1.1 无损压缩 ……423
8.1.2 有损压缩 ……431
8.2 多媒体数据 ……435
8.2.1 文本 ……435
8.2.2 图像 ……435
8.2.3 视频 ……438
8.2.4 音频 ……439
8.3 因特网中的多媒体 ……440
8.3.1 流式存储音频/视频 ……440
8.3.2 流式实况音频/视频 ……442
8.3.3 实时交互式音频/视频 ……443
8.4 实时交互式协议 ……447
8.4.1 新协议的基本原理 ……448
8.4.2 RTP ……450
8.4.3 RTCP ……452
8.4.4 会话初始化协议 ……454
8.4.5 H.323 ……459
8.4.6 SCTP ……460
8.5 服务质量 ……470
8.5.1 数据流量特征 ……470
8.5.2 流量分类 ……471
8.5.3 通过流量控制提高 QoS ……471
8.5.4 综合服务（IntServ）……475
8.5.5 区分服务（DiffServ）……478
8.6 章末资料 ……479
推荐读物 ……479
小结 ……480
8.7 习题集 ……480
测试题 ……480
练习题 ……480
思考题 ……482
8.8 模拟实验 ……487
Applets ……487
实验作业 ……487
8.9 编程作业 ……487

第 9 章 网络管理 ……488
9.1 介绍 ……488
9.1.1 配置管理 ……489
9.1.2 故障管理 ……490
9.1.3 性能管理 ……490
9.1.4 安全管理 ……491
9.1.5 计费管理 ……491
9.2 SNMP ……491
9.2.1 管理器和代理 ……491
9.2.2 管理组件 ……492
9.2.3 概要 ……493
9.2.4 SMI ……493
9.2.5 MIB ……497
9.2.6 SNMP ……499
9.3 ASN.1 ……502
9.3.1 语言的基本要素 ……503
9.3.2 数据类型 ……503
9.3.3 编码 ……505
9.4 章末资料 ……505
推荐读物 ……505
小结 ……506
9.5 习题集 ……506
测试题 ……506
练习题 ……506
思考题 ……507

第 10 章 网络安全 ……508
10.1 介绍 ……508
10.1.1 安全目标 ……508
10.1.2 攻击 ……509
10.1.3 服务和技术 ……510
10.2 机密性 ……511
10.2.1 对称密钥密码 ……511
10.2.2 非对称密钥密码 ……518
10.3 安全的其他方面 ……522
10.3.1 消息完整性 ……522
10.3.2 消息认证 ……523
10.3.3 数字签名 ……523
10.3.4 实体认证 ……527
10.3.5 密钥管理 ……529
10.4 Internet 安全 ……533
10.4.1 应用层安全 ……533
10.4.2 传输层安全 ……540
10.4.3 网络层安全 ……544
10.5 防火墙 ……551
10.5.1 分组过滤防火墙 ……552
10.5.2 代理防火墙 ……552
10.6 章末资料 ……553

推荐读物 ········· 553
小结 ········· 553
10.7 习题集 ········· 554
测试题 ········· 554
练习题 ········· 554
思考题 ········· 555
10.8 模拟实验 ········· 558
Applets ········· 558
实验作业 ········· 558
10.9 编程作业 ········· 558
第 11 章 Java Socket 编程 ········· 559
11.1 介绍 ········· 559
11.1.1 地址和端口 ········· 559
11.1.2 客户-服务器模式 ········· 562
11.2 UDP 编程 ········· 563
11.2.1 迭代方法 ········· 563
11.2.2 并发方法 ········· 571
11.3 TCP 编程 ········· 573
11.3.1 迭代方法 ········· 573
11.3.2 并发方法 ········· 581
11.4 章末资料 ········· 583
推荐读物 ········· 583
小结 ········· 583
11.5 习题集 ········· 583
测试题 ········· 583
练习题 ········· 583
思考题 ········· 584
11.6 编程作业 ········· 585
附录 A Unicode ········· 586
附录 B 按位计数系统 ········· 590
附录 C HTML、CSS、XML 和 XSL ········· 595
附录 D 其他信息 ········· 601
附录 E 8B/6T 编码 ········· 603
参考文献 ········· 605

HZ Books
华章图书

第 1 章

概　　论

最大的计算机网络——因特网（Internet），拥有超过 10 亿的用户。利用有线和无线传输介质，Internet 连接了大大小小的计算机系统。它允许用户共享包括文本、图像、音频和视频在内的大量信息，允许用户之间相互发送消息。本书的主要目标就是探索这个庞大的系统。在本章，我们有两个目标。第一个目标是对作为一个互联网（网中网）的 Internet 进行概述，讨论 Internet 的组成部分。这个目标的部分内容为介绍协议分层和 TCP/IP 协议簇。换句话说，第一个目标是为本书的其他章节做准备。第二个目标是提供相关的信息，但是这些信息在学习其他章节时不是必需的。

- 1.1 节介绍局域网（local area network，LAN）和广域网（wide area network，WAN），给出这两种类型网络的主要定义。我们定义一个局域网和广域网相结合的互联网络——互联网。我们将展示一个组织怎样利用广域网将它的局域网连接起来，从而创建一个私有的互联网。最后，我们介绍由主干、网络提供商和用户网络组成的 Internet，它是一个全球性的互联网络。
- 1.2 节我们利用协议分层（protocol layering）的概念展示 Internet 怎样将任务分解成多个小任务。我们将讨论 5 层协议簇（TCP/IP），介绍每层的任务和每层拥有的协议。我们还将讨论在这种模式下的两个概念：封装/解封装（encapsulation/decapsulation）和多路复用/多路分解（multiplexing/demultiplexing）。
- 1.3 节我们为感兴趣的读者介绍 Internet 的主要发展史。跳过这部分内容不会丧失本书的连续性。
- 1.4 节介绍 Internet 的管理、标准的制定和生命周期。这部分内容仅仅提供相关的信息，对理解本书其他章节内容不是必需的。

1.1　Internet 概览

尽管本书的目标是讨论 Internet，一个连接世界上数十亿计算机终端的系统，但是我们认为 Internet 不是一个单一的网络，而是一个互联网络（internetwork），多个网络的组合。所以，我们首先对网络进行定义，然后展示怎样连接多个网络来创建小型的互联网络。最后，我们介绍 Internet 的结构，进而打开后面 10 章学习 Internet 的大门。

1.1.1　网络

网络（network）是由一组具有通信能力的设备相互连接而形成的。在这个定义中，设备可以是主机（host，有时也称为*端系统*（end system），如大型计算机、桌面计算机、笔记本电脑、工作站、无线电话、安全系统等），也可以是连接设备，如连接网络到其他网络的路由器、将设备连接到一起的交换机、变换数据形式的调制解调器等。在一个网络中，这些设备使用有线或无线的传输介质（如电缆或大气）连接起来。当家庭中利用一台即插即用的路由器连接两台电脑时，我们就组建了一个网络，尽管这个网络很小。

局域网

局域网（local area network，LAN）通常是私有的，连接一个办公室、大楼或校园内的一些主机。按照需求的不同，一个局域网既可以简单地由两台电脑和一台打印机组成，用于家庭办公，也

HZ BOOKS 华章图书

可以贯穿整个公司，包含语音和视频设备。在局域网中的每台主机都具有一个标识符（一个地址），用于在局域网中唯一地定义这台主机。一台主机向另一台主机发送的数据包携带了源主机和目的主机的地址。

在过去，一个网络中的所有主机都连接到一个公共的电缆上，这意味着从一台主机发往另一台主机的数据包可以被所有的主机接收到。目标接收者保存这个数据包，而其他主机丢弃该数据包。现在，多数局域网采用智能连接交换机，它能够识别数据包的目的地址并引导该数据包到达它的目的地而不必将它发送到其他主机。交换机减轻了局域网中的流量，如果不是共同的源主机和目的主机，那么交换机允许同一时刻多对主机之间同时相互通信。注意，上面局域网的定义没有指定局域网中最小或最大的主机数。图 1-1 显示了使用公用电缆和交换机组成的局域网。

第 5 章和第 6 章将对局域网进行详细讨论。

当孤立地使用局域网时（目前已很少见），它们用于主机之间共享资源。我们马上能看到，目前的局域网常常相互连接，同时连接到广域网（下面我们将讨论），以进行更广范围的通信。

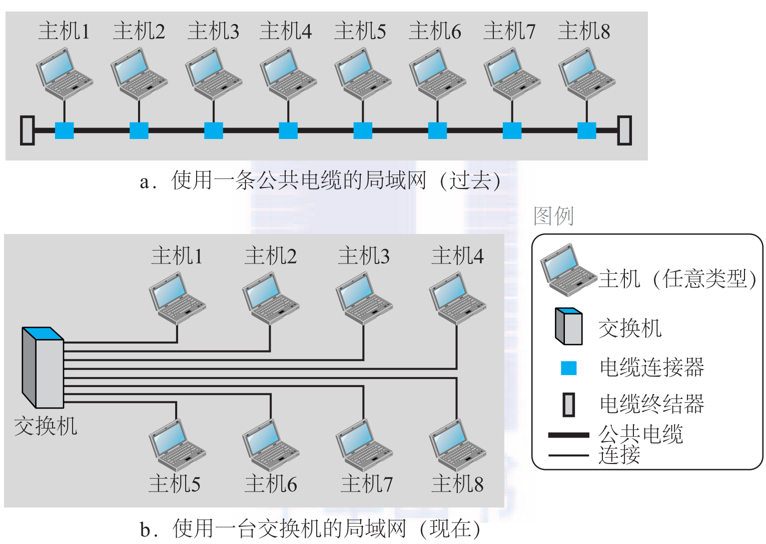


图 1-1　过去和现在的孤立局域网

广域网

广域网（wide area network，WAN）也是由具有通信能力的设备相互连接而形成的。可是，局域网和广域网有一些不同。局域网通常覆盖范围受限，可以覆盖一间办公室、一栋大楼或一个校园；广域网则具有更广的地理覆盖范围，可以覆盖一个城市、一个省、一个国家甚至整个世界。局域网互联主机；广域网互联交换机、路由器、调制解调器等连接设备。局域网通常由使用它的组织拥有；广域网通常由通信公司建设和运营，使用它的组织进行租用。我们看看目前使用的两种典型的广域网：点到点广域网和交换式广域网。

点到点广域网

点到点广域网通过传输介质（电缆或大气）连接两个通信设备。在讨论怎样把一个网络连接到另一个网络时，我们将看到这些广域网的例子。图 1-2 显示了一个点到点广域网的例子。

交换式广域网

交换式广域网具有多个端点。不久我们将看到，交换式广域网目前用作全球通信的主干。我们可以说，交换式广域网是交换机连接几个点到点的广域网而形成的。图 1-3 给出了一个交换式广域网

网的例子。



图 1-2 点到点广域网

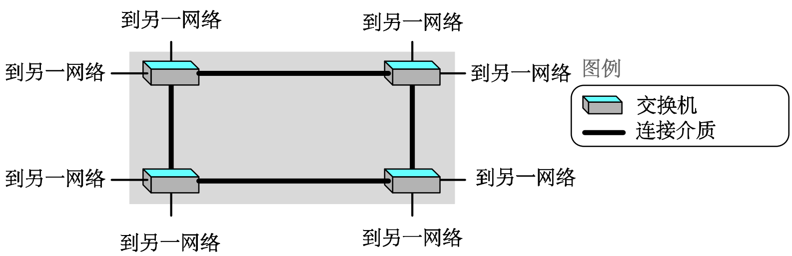


图 1-3 交换式广域网

第 5 章和第 6 章将对广域网进行详细讨论。

互联网络

现在我们很少看到孤立的局域网或广域网，它们都相互连接在一起。当两个或多个网络连接起来，它们就形成了一个互联网络（internetwork），或者互联网（internet）。例如，假如一个机构有两个办公室，一个在东海岸，另一个在西海岸。每一个办公室都拥有一个局域网，允许办公室的员工相互进行通信。为了使不同办公室的员工能够互相通信，管理部门从服务提供商（例如电话公司）租用一个点到点的专用广域网，用来连接两个局域网。现在公司拥有了一个互联网络，或者说一个私有互联网。不同办公室之间的通信变成了可能。图 1-4 显示了这个互联网。

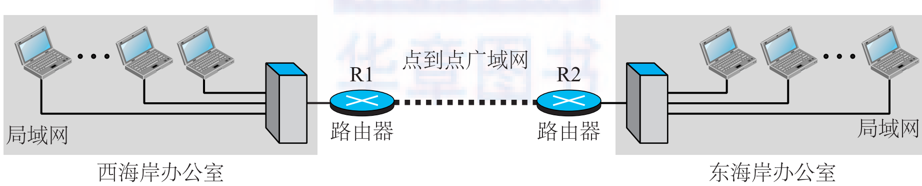


图 1-4 由两个局域网和一个点到点广域网组成的互联网络

当西海岸办公室中的一台主机给同一办公室的另一台主机发送消息时，路由器阻截这条消息，但是交换机指引这条消息到达目的地。另一方面，当西海岸的一台主机给东海岸的一台主机发送消息时，路由器 R1 将数据包路由到路由器 R2，然后数据包到达目的地。

图 1-5 显示了多个局域网和广域网连接形成的另一个互联网。广域网之一为具有 4 个交换机的交换式广域网。

1.1.2 交换

我们已经说过，互联网是由链路和交换机组成的，例如我们前面使用的链路层交换机和路由器。实际上，互联网是一个交换式的网络，其中一台交换机至少将两条链路连接在一起。当需要的时候，交换机需要将数据从一条链路转发到另一条链路。交换式网络最常见的类型为电路交换网络和分组交换网络。下面我们讨论这两种类型的网络。

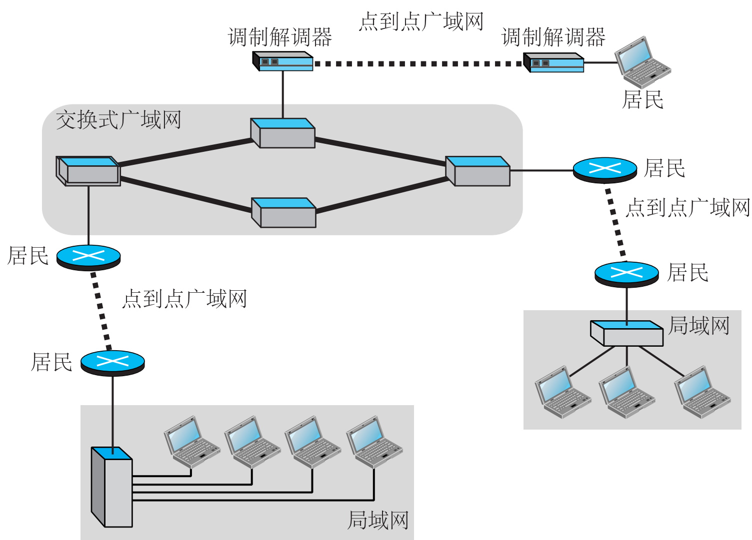


图 1-5　一个由 4 个广域网和 3 个局域网组成的异构网络

电路交换网络

在**电路交换网络**（circuit-switched network）中，两个端系统之间总是存在一条专用的连接（称为电路），交换机只能使其变成活跃或非活跃状态。图 1-6 显示了一个简单的交换式网络，该网络在每端连接 4 部电话。由于过去电话网络经常采用电路交换，因此我们使用电话机代替计算机作为端系统，尽管目前部分电话系统采用分组交换网络。

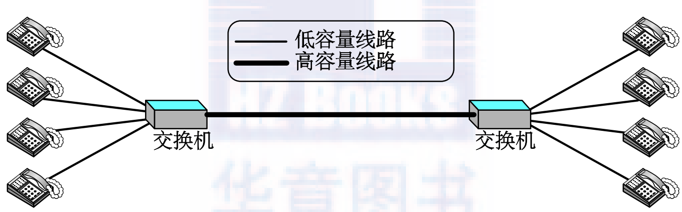


图 1-6　电路交换网络

在图 1-6 中，每端的 4 部电话连接到一个交换机。交换机将一端的电话机连接到另一端的电话机。连接两台交换机的粗线是一个高容量的通信线路，它能够同时处理 4 路语音通信，其容量能够被所有电话对之间共享。本例使用的交换机具有转发功能但是没有存储能力。

我们看看以下两种情况。在第一种情况下，所有电话机均处于忙状态；一端的 4 个人正在与另一端的 4 个人进行通话；粗线的容量被完全使用。在第二种情况下，一端只有一部电话机连接到另一端的电话机；粗线容量仅仅四分之一被使用。这意味着仅当占用全部容量时，电路交换网络才具有高效率；在多数时间中，由于工作仅仅占用部分容量，因此它的效率低下。需要将粗线的容量做成每条语音线路容量 4 倍的原因是，当一端的所有电话机想要与另一端所有电话机连接时，我们不希望通信失败。

分组交换网络

在一个计算机网络中，两个端点之间使用被称为**分组**（packet）的数据块进行通信。也就是说，与正在使用的电话机之间连续通信不同，两台计算机之间交换的是独立的数据分组。由于分组是一个能够被存储和以后发送的独立实体，因此这种机制允许我们实施存储转发的交换功能。图 1-7 显示了一个每端分别连接 4 台计算机的小型分组交换网络。

分组交换网络中的路由器具有能够存储和转发分组的队列。现在假设粗线的容量（即高容量）

HZ BOOKS 华章图书

仅仅为连接计算机到路由器数据线容量的两倍。如果只有两台计算机（分别在两端）需要相互通信，那么发送的分组不需要等待。但是，如果当粗线已经工作在满负荷时分组到达一个路由器，那么应该存储分组并且按照它们到达的次序进行转发。虽然这两个简单的例子显示分组交换网络比电路交换网络效率高，但是分组可能会遇到一些延迟。

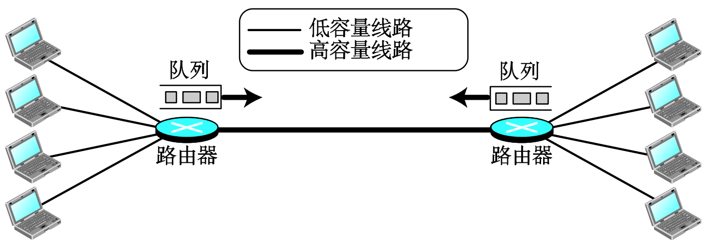


图 1-7　分组交换网络

本书我们主要讨论分组交换网络。在第 4 章中，我们将详细讨论分组交换网络，同时讨论这些网络的性能。

1.1.3　Internet

正像我们以前讨论的那样，互联网是由两个或多个能够相互通信的网络组成的。最著名的互联网叫做因特网（Internet），Internet 由成千上万个相互连接的网络组成。图 1-8 显示了一个 Internet 概念上（而不是地理上）的视图。

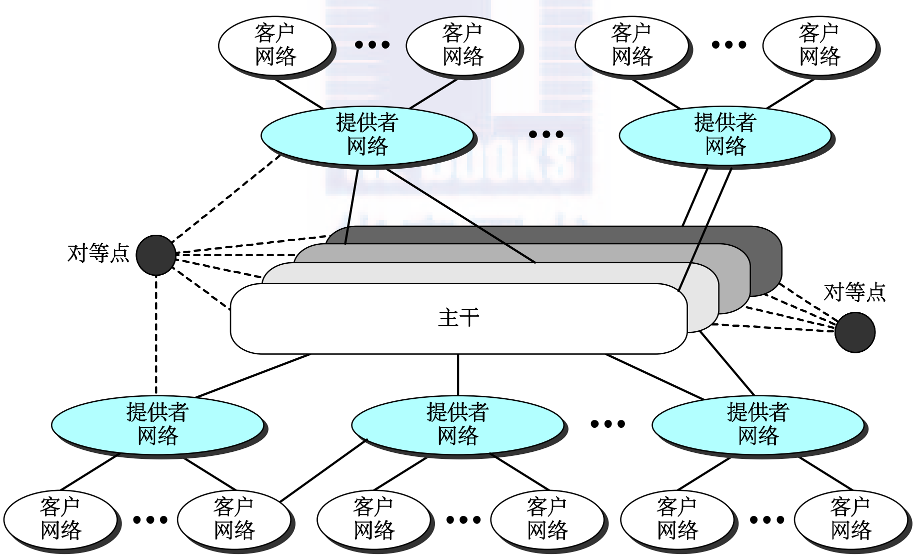


图 1-8　现今的 Internet

从图 1-8 中看到，Internet 由一系列主干、提供者网络和客户网络组成。主干（backbone）处于最高层次，是一些通信公司拥有的大型网络，如 Sprint、Verizon(MCI)、AT&T、NTT。主干网络通过称为对等点（peering point）的复杂交换系统进行连接。一些小些的提供者网络（provider network）处于第二个层次，这些网络通过付费使用主干网络服务。提供者网络连接主干网络，有时提供者网络之间也相互连接。客户网络（customer network）是 Internet 边缘的网络，它们使用 Internet 提供的服务。为了接收服务，客户网络需要向提供者网络付费。

主干和提供者网络也称为 **Internet 服务提供商**（Internet Service Provider，ISP）。主干常常称为

国际 ISP；提供者网络常常称为国家或区域 ISP。

1.1.4　访问 Internet

今天的 Internet 是一个允许任何用户变成它的一部分的互联网。但是，用户需要物理上连接到一个 ISP。物理连接通常利用一条点到点的广域网实现。本节我们会简单描述怎样进行连接，至于连接的详细技术信息我们将在第 6 章和第 7 章讨论。

使用电话网络

目前大多数居民和小公司具有电话服务，这就意味着他们能够连接到电话网络。由于大多数电话网络自身已经连接到 Internet，因此居民和小公司连接 Internet 的一个选择是把他们和电话中心的语音线路转换成点到点的广域网。这可以用两种方式实现。

- **拨号服务**。第一种解决方法是在电话线路中增加将数据转换成语音的调制解调器。安装在计算机中的软件拨打 ISP 的号码，形成一条电话连接。非常不幸，拨号服务非常慢，同时当线路用于 Internet 连接时，线路就不能进行电话（语音）连接。因此，这种方式只对偶尔访问 Internet 的居民和小公司有效。我们将在第 5 章中讨论拨号服务。
- **DSL 服务**。自从 Internet 出现后，一些电话公司开始升级它们的电话线路，以向居民和小公司提供较高速率的 Internet 服务。DSL 服务允许语音和数据通信同时进行。我们将在第 5 章讨论 DSL。

利用有线电视网络

近 20 年，越来越多的居民开始使用有线电视服务替代天线接收电视广播。有线电视公司已经升级了它们的有线电视网络并连接到 Internet。居民和小公司可以通过这种服务连接到 Internet。虽然这种方法可提供较高速率的连接，但是速率与使用同一电缆的用户数目有关。我们将在第 5 章讨论有线电视网络。

采用无线网络

无线连接最近变得非常流行。住户或小公司可以使用无线和有线连接混合的方法访问 Internet。随着无线广域网接入的发展，住户或小公司能够通过无线广域网连入 Internet。我们将在第 6 章讨论无线接入。

直接连接到 Internet

大机构或大公司自身可以变成一个本地 ISP 并连入 Internet。这种方法要求组织或公司从一个线路提供者那里租用高速广域网并将它连入地区 ISP。例如，具有几个校园的大学可以组建一个互联网，然后连接互联网至 Internet。

1.1.5　硬件和软件

我们已经给出了 Internet 结构的概览，Internet 是由连接设备将大型和小型网络连接起来形成的。但是应该清楚地看到，仅仅连接这些东西是不够的。为了使通信正常进行，我们既需要硬件也需要软件。这就像一个复杂的计算，我们既需要计算机也需要程序。在下一节，我们讨论如何利用协议分层对硬件和软件的组合进行相互协调。

1.2　协议分层

当谈到 Internet 时，我们总能听到的一个词汇就是**协议**（protocol）。协议定义了发送者、接收者和所有中间设备为了高效通信需要遵循的规则。当通信简单时，我们可能只是需要一个简单的协议；当通信复杂时，我们可能需要把任务划分到不同层，每层需要一个协议，也就是说需要**协议分层**（protocol layering）。

1.2.1　场景

为了更好地理解为什么需要协议分层，我们来看两种简单的场景。

场景一

在第一个场景中，通信如此简单以至于它能够在一层中实现。假设 Maria 和 Ann 是拥有很多共同想法的邻居。Maria 和 Ann 之间的通信发生在一个层次中，她们面对面并使用相同的语言，如图 1-9 所示。



图 1-9　单一层次的协议

即使在这个简单的场景中，我们也可以看到需要遵循一系列的规则。第一，Maria 和 Ann 了解当她们相遇时应该互相问候。第二，她们明白她们应该限制使用的词汇在她们友谊的层次上。第三，一方知道当另一方讲话时，她应该抑制自己讲话。第四，每一方都知道应该是对话而不是独角戏：双方都应该有机会对某一问题发表看法。第五，当她们分别时，应该交换一些祝福语。

我们可以看到，Maria 和 Ann 使用的协议与课堂上老师和学生的通信不同。第二种情况中，大部分情况下是独角戏；除非学生有问题，老师的谈话会占用大部分时间。在这种情况下，协议应该规定学生应该举手并等待被允许说话。这种情况下的通信通常非常正式，同时限制在讲授的前提下。

场景二

在第二个场景中，我们假设公司提拔了 Ann，但是她需要到距离 Maria 很远的城市上班。由于这两位朋友着手进行一个新的项目以便退休后启动新生意，因此她们希望继续她们的通信、交换她们的想法。她们决定定期通过邮局使用信件继续交换她们的想法。但是，即使信件被拦截，她们也不希望她们的想法被其他人知道。她们一致同意采用一种加密/解密技术。信件的发送者对信件加密，使之对入侵者不可读；信件的接收者对信件解密，从而得到原始的信件。我们将在第 10 章讨论加密/解密方法，但是现在我们假设 Maria 和 Ann 采用了一种技术，该技术在一个人不拥有密钥的情况下很难解密信件。现在我们可以说 Maria 和 Ann 之间的通信在 3 个层次上进行，如图 1-10 所示。我们假设 Ann 和 Maria 每个人拥有 3 台机器（机器人），这 3 台机器分别在每一层执行任务。

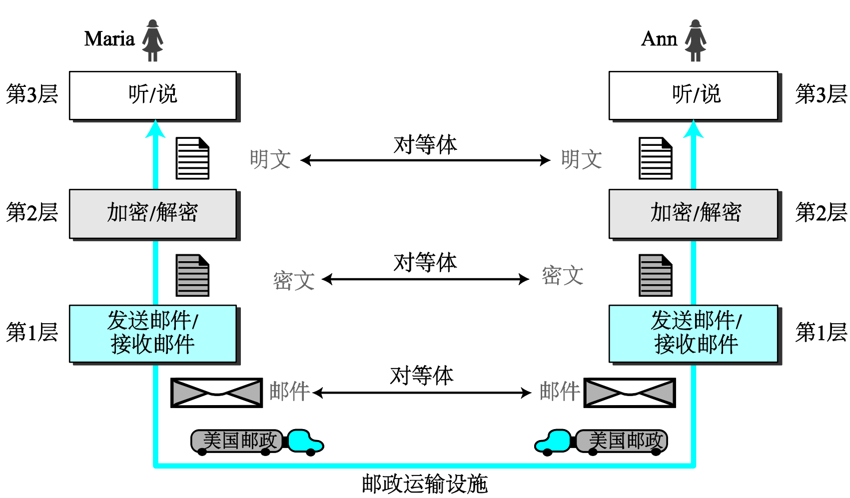


图 1-10　一种三层协议

我们假设 Maria 向 Ann 发送第一封邮件。Maria 在第 3 层对机器谈话，仿佛这台机器就是 Ann，并且在听她谈话。第 3 层机器聆听 Maria 所说的内容并形成了明文（用英文书写的邮件），传递给第 2 层机器。第 2 层机器接收明文，对它进行加密并形成密文，传递给第 1 层机器。第 1 层机器，大概是个机器人，接收密文，把它放入一个信封中，添加发送者和接收者地址，然后进行邮寄。

在 Ann 的一端，第 1 层机器从 Ann 的信箱中取出邮件，通过发送者的地址得知该邮件来自于 Maria。机器从信封中取出密文并将它投递给第 2 层机器。第 2 层机器解密这个信息，形成明文并将明文传递给第 3 层机器。第 3 层机器接收明文并仿佛 Maria 正在说话一样将它读出来。

协议分层允许我们将一个复杂的任务分解成几个较小的、简单的任务。例如，在图 1-10 中，我们可以只使用一台机器完成所有 3 台机器的工作。可是，如果 Maria 和 Ann 判定这台机器所做的加密/解密不足以保护她们的秘密，那么她们需要更换整台机器。在现在的情况下，她们只需要更换第 2 层的机器，另外两台机器能够保持不变。这种方法称为模块化（modularity）。在这个示例中，模块化意味着独立的层次。一层（一个模块）可以定义为一个具有输入和输出的黑盒子，我们不必关心输入如何变成输出。如果给定相同的输入，两台机器提供相同的输出，那么它们可以相互替换。例如，Ann 和 Maria 可以从两个不同的厂商购买第 2 层机器。只要这两台机器能把相同的明文变成相同的密文，相同的密文变成相同的明文，那么它们就可以相互替换。

协议分层的优越性之一是它允许我们将服务从实现中分离出来。一层需要能够接收较低层的一系列服务，同时向较高层提供服务，而我们不关心这一层是如何实现的。例如，Maria 可以决定不为第 1 层购买机器（机器人）；她可以自己做这些工作。只要 Maria 能够在两个方向上完成第 1 层提供的任务，通信系统就可以工作。

协议分层的另一个优越性无法在简单的示例中体现，但是当我们讨论 Internet 中的协议分层时能够展现出来。这个优越性就是通信不只是用于两个端系统，中间系统只需要一些层次而不是所有的层次。如果不使用协议分层，形成的中间系统就不得不像端系统一样复杂，这样就会提高整个系统的造价。

协议分层有不足之处吗？有人说单一层次使工作更加容易。对每一层来说，没有必要向上一层提供服务并使用下一层的服务。例如，Ann 和 Maria 可以寻找或建造一台机器来完成这三种任务。可是，正像前面提到的那样，如果某一天她们发现她们的编码被攻破，那么每人都不得不用新机器替换整个机器，而不是只更换第 2 层的机器。

协议分层原则

让我们讨论一些协议分层原则。第一个原则就是如果想要双向通信，那么我们需要每一层能够实现两个相反的任务，每个方向上一个。例如，第 3 层的任务就是听（在一个方向上）和说（在另一个方向上），第 2 层需要能够加密和解密，第 1 层需要发送和接收邮件。

在协议分层中我们需要遵循的第二个原则是两端每一层中的两个对象应该相同。例如，两端第 3 层的对象应该为明文信件。两端第 2 层的对象应该为密文信件。两端第 1 层的对象应该为一封邮件。

逻辑连接

在遵循以上两个原则之后，每层之间的逻辑连接如图 1-11 所示。这意味着我们拥有层到层的通信。Maria 和 Ann 可以认为每一层有一个逻辑（想象的）连接，通过这个连接她们可以发送那一层创建的对象。我们将看到逻辑连接的概念将帮助我们更好地理解数据通信与网络中遇到的分层任务。

1.2.2　TCP/IP 协议簇

在第 2 个场景中，我们了解了协议分层与层间逻辑连接的概念，现在介绍 TCP/IP（Transmission Control Protocol/Internet Protocol，传输控制协议/互联网协议）。TCP/IP 是目前 Internet 使用的一个协议簇（按不同层次组织的协议集）。它是由相互交互的模块组成的一个层次结构协议，每一个模块提供特定的功能。*层次*意味着较上层次的协议需要得到一个或多个较下层次协议提供的服务支持。初始的 TCP/IP 协议簇在硬件基础上定义了 4 个软件层次。但是，目前 TCP/IP 通常是一个 5

层模型。图 1-12 显示了这两种情况。

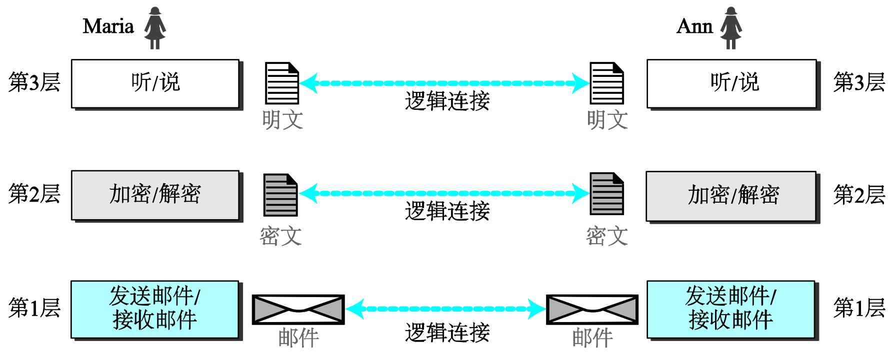


图 1-11　对等层之间的逻辑连接

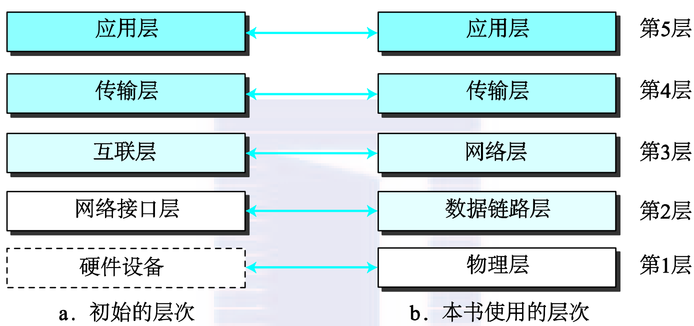


图 1-12　TCP/IP 协议簇的层次

层次化结构

为了展示如何利用 TCP/IP 协议簇的层次在两台主机之间进行通信，我们假设将协议簇用于一个由 3 个局域网（链接）组成的小型互联网，每个局域网拥有一个链路层交换机。同时我们假设局域网连接到一个路由器，如图 1-13 所示。

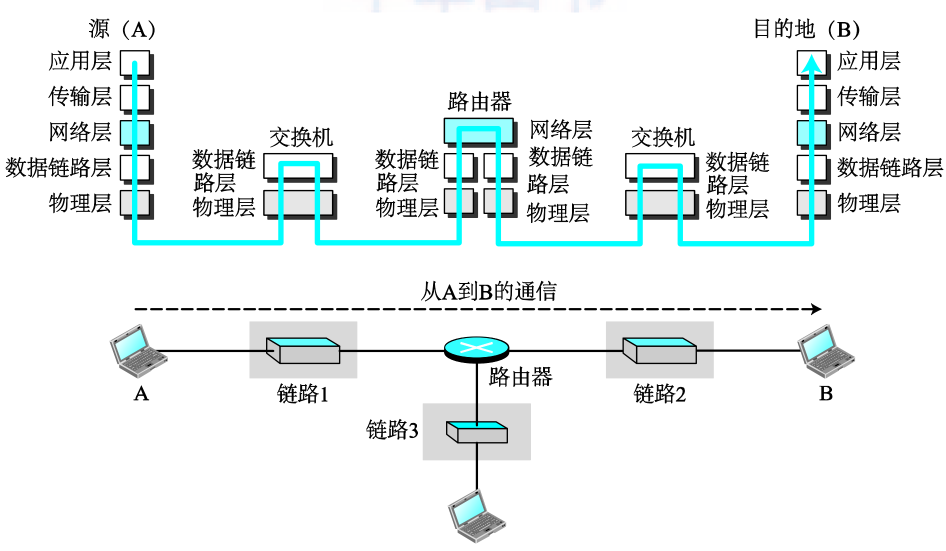


图 1-13　通过一个互联网通信

我们假设计算机 A 与计算机 B 进行通信。正像图 1-13 中显示的那样，在这个通信中涉及 5 个通信设备：源主机（计算机 A）、链路 1 的链路层交换机、路由器、链路 2 的链路层交换机和目的主机（计算机 B）。按照设备在互联网中扮演的角色不同，每一台设备包含了几个层次。两台主机包含了所有 5 个层次；源主机需要在应用层创建一个信息并把它发送到下层，以便把该信息物理上发送到目的主机。目的主机需要在物理层接收这个信息，然后通过其他层投递到应用层。

路由器只涉及 3 层；只要路由器仅仅作为路由选择，在路由器中就没有传输层或应用层。虽然一个路由器总是拥有一个网络层，但是它涉及 *n* 个数据链路层和物理层的组合，其中 *n* 为路由器连接的链路的数目。其主要原因是每一个链路可以使用它自己的数据链路或物理层。例如在图 1-13 中，路由器拥有 3 条链接，但是从源 A 发送到目的地 B 的消息涉及两条链接。每一条链接可以使用不同的链路层和物理层协议；路由器需要从基于一对协议的链路 1 接收分组并将它投递到基于另一对协议的链路 2。

可是，在一条链路上的链路层交换机只涉及两个层次：数据链路层和物理层。尽管图 1-13 显示的交换机拥有两个不同的连接，但是这两个连接在同一链路上，它们只使用一个协议集。这意味着链路层交换机与路由器不同，它只涉及一个数据链路层和一个物理层。

TCP/IP 协议簇中的层次

在进行前面的介绍之后，我们来简单讨论 TCP/IP 协议簇中层次的功能和任务。接下来的 6 章将对每一层进行详细讨论。为了更好地理解每一层的任务，我们首先需要知道在层次间存在的逻辑连接。图 1-14 显示了简单互联网的逻辑连接。

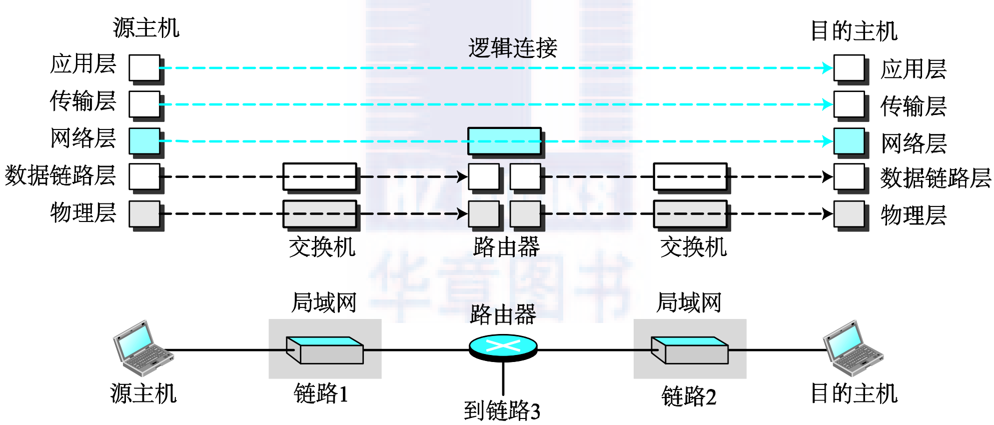


图 1-14　TCP/IP 协议簇中层次间的逻辑连接

采用逻辑连接使我们考虑每一层的任务变得比较容易。如图 1-14 所示，应用层、传输层和网络层的任务是端到端的（end-to-end）。但是，数据链路层和物理层的任务是点到点的（hop-to-hop），其中一个跳步是一个主机或路由器。也就是说，高三层的任务范围是互联网，低两层的任务范围是链路。

另一种理解逻辑连接的方法是考虑每一层创建的数据单元。在高三层，数据单元（分组）不应该被任何路由器或链路层交换机改变。在低两层，主机创建的分组仅仅被路由器改变，链路层交换机不改变它们。

图显示了前面讨论的协议分层的第二个原则。我们看一看与设备相关的每一层之下的对等体。

注意，尽管网络层的逻辑连接在两个主机之间，但是由于一个路由器在网络层对分组进行分片，并且发送的分组比接收的多（见第 4 章分片部分），因此在这种情况下，对等体只存在于两个跳步之间。

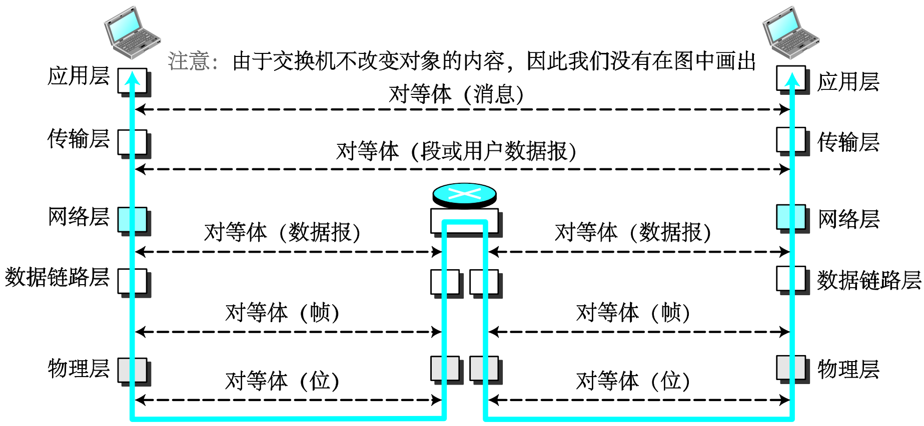


图 1-15　TCP/IP 协议簇中的对等体

TCP/IP 各层描述

理解逻辑通信的概念之后，我们简单讨论各层的主要任务。本章的讨论将非常简单，不过我们会在接下来的 6 章继续讨论各层的功能与任务。

应用层

如图 1-14 所示，两个应用层之间的逻辑连接是端到端的。两个应用层之间仿佛存在一座桥梁一样相互交换消息。可是，我们应该明白通信需要通过所有层次完成。

应用层的通信处于两个进程（该层正在运行的两个程序）之间。为了进行通信，一个进程向另一个进程发送请求，并且接收另一个进程的响应。进程到进程的通信就是应用层的任务。虽然 Internet 的应用层包含了很多预定义的协议，但是也可以在两台主机上运行用户创建的一对进程。我们将在第 2 章研究这种情况。

超级文本传输协议（Hypertext Transfer Protocol，HTTP）是访问万维网（World Wide Web，WWW）的载体。简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）是电子邮件（e-mail）服务的主要协议。文件传输协议（File Transfer Protocol，FTP）用于将文件从一台主机传输到另一台主机。远程登录（Terminal Network，TELNET）和安全外壳（Secure Shell，SSH）用于访问远端的站点。管理员使用简单网络管理协议（Simple Network Management Protocol，SNMP）对 Internet 全局或局部进行管理。域名系统（Domain Name System，DNS）使其他的协议能够查询一台计算机的网络层地址。因特网组管理协议（Internet Group Management Protocol，IGMP）用于管理一个组的成员资格。我们将在第 2 章讨论这些协议的大部分，其他协议在另外一些章节讨论。

传输层

传输层的逻辑连接也是端到端的。源主机的传输层从应用层得到消息，封装成传输层的分组（称为段或用户数据报，不同协议叫法不同），然后进行发送。通过逻辑（想象的）连接，分组到达目的主机的传输层。也就是说，传输层负责向应用层提供服务：从运行于应用层的程序得到信息，并将它投递到目的主机相应的应用程序。我们也许要问为什么我们已经拥有了一个端到端的应用层，还需要端到端的传输层。与我们前面讨论的一样，其主要原因是分割任务与责任。传输层应该独立于应用层。另外，我们将看到传输层有多个协议，这意味着每个应用程序可以使用与它的需求最匹配的协议。

正像我们说的，Internet 中有几个传输层协议，每个都是为一些特定的任务设计的。作为主要的协议，传输控制协议（Transmission Control Protocol，TCP）是一个面向连接的协议，它在传输数据之前，首先在两台主机的传输层之间建立一条逻辑连接。TCP 协议在两个 TCP 层之间创建一

个管道，以便传输字节流。TCP 协议提供流量控制（匹配源主机的发送数据速率与目的主机的接收数据速率，以防止目的主机溢出）、差错控制（保证数据段无差错到达目的地和重新发送受损的数据段）、拥塞控制（减少由于网络拥塞造成的数据段丢失）。另一种常用的协议是用户数据报协议（User Datagram Protocol，UDP）。UDP 是一种无连接协议，它传输用户数据报之前不需要创建逻辑连接。在 UDP 中，每个用户数据报是一个独立的实体，它和前一个或后一个用户数据报没有关系（无连接就是这个意思）。UDP 是一种比较简单的协议，它不提供流量控制、差错控制或拥塞控制。它的简单性（意味着小的额外开销）对某些应用程序具有吸引力，这些应用程序发送较短的消息且不能容忍 TCP 在分组损坏或丢失时使用重发机制。流控制传输协议（Stream Control Transmission Protocol，SCTP）是一种新协议，它是为多媒体出现的新应用设计的。我们将在第 3 章讨论 UDP 和 TCP，第 8 章讨论 SCTP。

网络层

网络层负责在源计算机和目的计算机之间创建一个连接。网络层的通信是主机到主机的。可是，由于从源主机到目的主机可能存在多个路由器，因此路径上的路由器负责为每个分组选择最好的路径。我们可以说网络层负责主机到主机的通信，并且指挥分组通过合适的路由器。我们再次问一问我们自己为什么需要网络层。我们可以在传输层增加路由任务，同时去掉这一层。正像我们前面介绍的，原因之一是在不同的层次之间分割不同的任务。原因之二是路由器不需要应用层和传输层。分割任务允许我们在路由器上加载较少的协议。

Internet 的网络层包括其主要协议：因特网协议（Internet Protocol，IP），因特网协议定义了在网络层称为数据报的分组格式。IP 同时定义了在这一层使用的地址格式和结构。与此同时，IP 负责从源主机把一个分组路由到目的主机。这种功能主要是通过每个路由器都将数据报转发到路径上的下一个路由器而实现的。

IP 是一个无连接的协议，不提供流量控制、差错控制和拥塞控制服务。这意味着如果一个应用需要这些服务，那么应用需要依赖于传输层协议。网络层也包括单播（一对一）和多播（一对多）路由协议。虽然路由协议不参加路由（路由是 IP 的责任），但是它为路由器创建转发路由表，为转发处理提供帮助。

网络层也包含一些帮助 IP 转发和进行路由工作的辅助协议。在路由一个分组时，因特网控制报文协议（Internet Control Message Protocol，ICMP）帮助 IP 报告遇到的问题。因特网组管理协议（Internet Group Management Protocol，IGMP）协助 IP 进行多任务处理。动态主机配置协议（Dynamic Host Configuration Protocol，DHCP）帮助 IP 获取一台主机的网络层地址。在网络层地址已知时，地址解析协议（Address Resolution Protocol，ARP）帮助 IP 寻找一台主机或一台路由器的链路层地址。我们在第 4 章讨论 ICMP、IGMP 和 DHCP，在第 5 章讨论 ARP。

数据链路层

我们已经知道一个互联网是多个链路（LAN 和 WAN）通过路由器连接而构成的。从主机传输数据报到目的地可能存在多个交叠的链路集。路由器负责选择最好的链路进行传输。可是，当路由器定好需要传输的下一条链路后，数据链路层接管这个数据报并使它穿过这条链路。这条链路可以是一个具有链路层交换机的有线局域网、一个无线局域网、一个有线广域网或者一个无线广域网。对于不同链路类型也存在不同的协议。无论哪种情况，数据链路层都要负责通过链路传输分组。

TCP/IP 没有为数据链路层定义任何特定的协议。它支持所有标准的和私有的协议。能够接管数据报并携带它穿过链路的任何协议都能满足网络层的要求。数据链路层接管一个数据报并将它封装在一个称为帧（frame）的分组中。

每个链路层协议可能提供不同的服务。有些链路层协议提供完整的检查和纠错，有些只提供纠错。我们在第 5 章讨论有线链路，在第 6 章讨论无线链路。

物理层

我们可以说物理层负责携带一个帧中单独的比特穿过链路。尽管物理层位于 TCP/IP 协议簇的最底层，但是由于在物理层之下存在另外一个隐藏的传输介质层，因此两个设备物理层之间的通信仍然是逻辑通信。两个设备通过一种传输介质（电缆或大气）连接。我们需要知道传输介质不携带比特；它携带电或光信号。所以，从数据链路层接收的一个帧的比特需要被变换，然后通过传输介质传输。但是我们可以认为两个设备物理层之间的逻辑单元是一个比特（bit）。将一个比特变换成一个信号存在多种协议。我们将在第 7 章讨论物理层和传输介质时讨论这些内容。

封装和解封装

在 Internet 协议分层中，一个重要的概念是封装/解封装。图 1-16 显示了图 1-13 给出的小型互联网的封装/解封装情况。

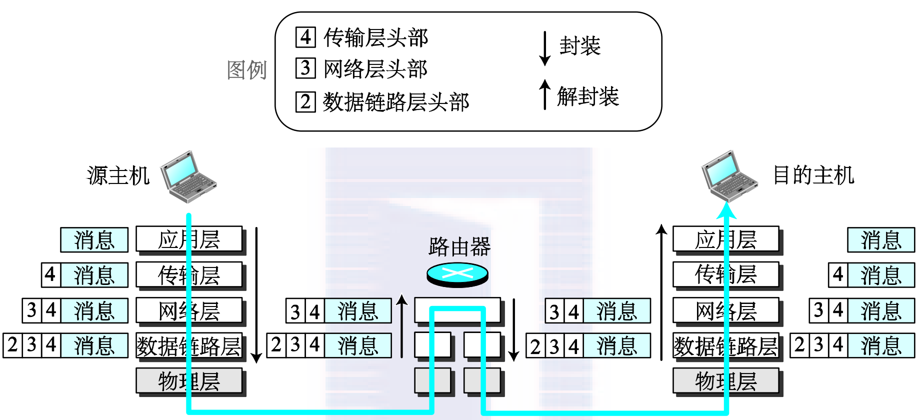


图 1-16　封装/解封装

由于在链路层交换机中没有封装/解封装发生，因此我们没有显示链路层交换机的层次。在图 1-16 中，我们显示了源主机中的封装、目的主机的解封装，以及路由器的封装和解封装。

源主机的封装

在源主机端，我们只进行封装。

1．在应用层，交换的数据称为消息（message）。消息通常不包含任何头部和尾部，但是即使包含了这些，我们也将其整体称为消息。消息会被传递到传输层。

2．传输层把这个消息作为有效载荷，该载荷是传输层应该关注的负载。传输层在有效载荷基础上增加传输层头部，其中包括了希望进行通信的源和目的应用程序的标识符和一些投递该消息需要的更多信息，例如进行流量控制、差错控制和拥塞控制需要的信息。其结果为一个传输层分组。该分组在 TCP 中称为段（segment），在 UDP 中称为用户数据报（user datagram）。然后传输层传递该分组到网络层。

3．网络层把传输层分组作为数据或有效载荷，并且在该有效载荷上添加自己的头部。头部包含源和目的主机的地址，以及用于头部差错检查、分片的信息等其他一些信息。其结果为一个称为数据报（datagram）的网络层分组。然后，网络层传递这个分组到数据链路层。

4．数据链路层把网络层分组作为数据或有效载荷，并且添加上自己的头部。该头部包含主机或下一跳步（路由器）的链路层地址。其结果为一个称为帧（frame）的链路层分组。该帧被传递到物理层进行传输。

路由器的解封装与封装

由于路由器连接两个或多个链路，因此在路由器中我们既需要进行解封装也需要进行封装。

1．在比特集被投递到数据链路层后，这一层从帧中解封装出数据报并将它投递到网络层。

2．网络层只检查数据报头部的源地址和目的地址，查阅它的转发表以寻找该数据报将被投递到的下一跳步。除非数据报太大以至于不能通过下一链路时需要对其进行分片，数据报的内容不应该被网络层改变。然后，数据报被传递到下一链路的数据链路层。

3．下一链路的数据链路层将数据报封装成一个帧，将其传递到物理层进行传输。

目的主机的解封装

在目的主机端，每层都只解封装接收到的分组，移出有效载荷，并将有效载荷传递至较高一层，直到消息到达应用层。需要说明的是主机中的解封装包含差错检查。

地址

在 Internet 中，与协议分层相关的另一个概念是地址。正像以前讨论的那样，在这种模型中一对层次之间存在逻辑通信。包含两方的任意通信都需要两个地址：源地址和目的地址。尽管看起来我们似乎需要 5 对地址（每层一对），但是由于物理层不需要地址，我们通常只需要 4 对；物理层的数据交换单元是一个比特，它绝对没有地址。图 1-17 显示了每一层的地址。

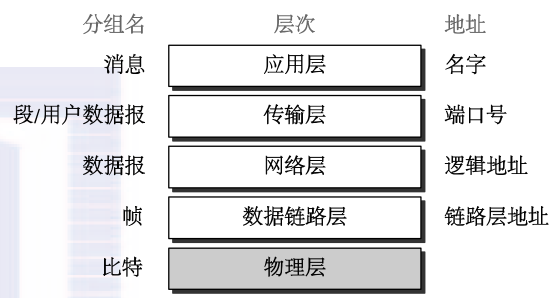


图 1-17　TCP/IP 协议簇中的地址

如图 1-17 所示，层次、地址与分组名之间存在一定的关系。在应用层，我们通常使用一个像 someorg.com 的名字定义提供服务的站点，或者使用像 somebody@coldmail.com 一样的电子邮件地址。在传输层，地址称为端口号，这些端口号指定源和目的地的应用层程序。端口号是本地地址，用于区分同一时间运行的几个程序。网络层地址是全局的，其范围涵盖了整个 Internet。链路层地址有时叫做 MAC 地址（MAC address），是本地定义的地址。每个链路层地址用于在网络（LAN 或 WAN）中定义一个特定的主机或路由器。在后面的章节中，我们将回过头来讨论这些地址。

多路复用与多路分解

由于 TCP/IP 协议簇在一些层次使用多个协议，因此我们在源端需要进行多路复用（multiplexing），在目的端需要进行多路分解（demultiplexing）。在这种情况下，多路复用的意思是一个协议能够封装来自多个上层协议的分组（一次一个）；多路分解的意思是一个协议能够进行解封装，并且将分组投递到多个上层协议（一次一个）。图 1-18 显示了一个高三层的多路复用与多路分解。

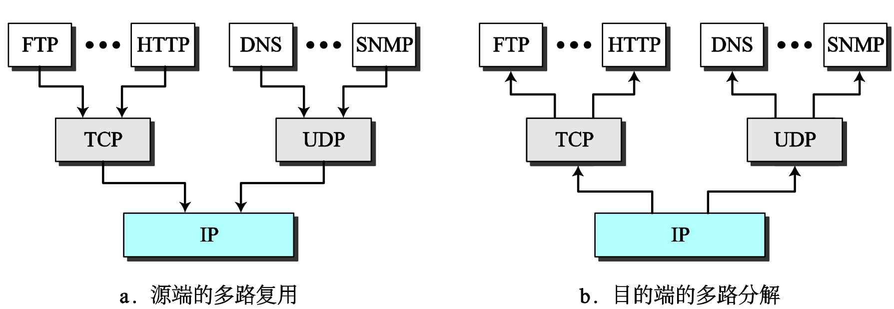


图 1-18　多路复用与多路分解

为了进行多路复用和多路分解，协议需要一个用于标识被封装的分组属于哪种协议的头部字段。在传输层，无论 UDP 还是 TCP 都可以接收多个应用层协议的消息。在网络层，IP 既可以接收来自 TCP 的段也可以接收来自 UDP 的用户数据报。同时，IP 也可以接收来自其他协议的分组，如

ICMP 协议、IGMP 协议等等。在数据链路层，数据帧可以携带来自 IP 或 ARP（参见第 5 章）等协议的有效载荷。

1.2.3　OSI 模型

虽然人们说起 Internet 都会谈到 TCP/IP 协议簇，但这个协议簇不是唯一被定义的协议簇。创建于 1947 年的**国际标准化组织**（International Organization for Standardization，ISO）由多个国家的成员组成，致力于世界范围内国际标准的制定。世界上将近四分之三的国家参加了 ISO 组织。**开放系统互连**（Open Systems Interconnection，OSI）模型是一个覆盖网络通信所有方面的 ISO 标准。它的第一次提出是在 20 世纪 70 年代末。

> ISO 是一个组织；OSI 是一种模型。

开放系统（open system）是一个允许任意两个不同系统进行通信的协议集，无论这两个系统采用何种结构。OSI 模型的主要目标是给出在不改变底层硬件和软件逻辑的情况下，怎样更利于不同系统之间的通信。OSI 模型不是一种协议；它是理解和设计具有可扩展性、健壮性和互操作性网络结构的模型。OSI 模型的目的是作为创建 OSI 协议栈的基础。

OSI 模型是设计网络系统的层次化架构，它允许所有类型计算机系统之间通信。它包含 7 个隔离但又相关的层次，每一层定义通过网络传输信息的一部分工作（如图 1-19 所示）。

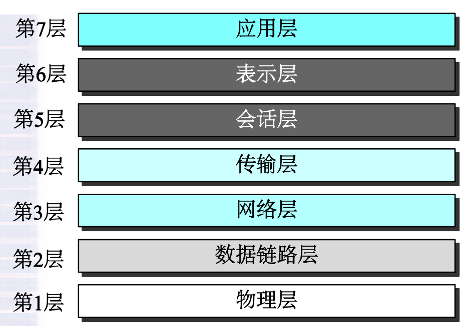


图 1-19　OSI 模型

OSI 与 TCP/IP

当比较 OSI 与 TCP/IP 时，我们发现 TCP/IP 协议簇没有会话层和表示层。在 OSI 模型发布后，这两层没有加入 TCP/IP 协议簇中。一般认为 TCP/IP 协议簇的应用层包含了 OSI 模型的 3 层，如图 1-20 所示。

对于这种决定，提到的原因有两点。首先，TCP/IP 存在一种以上的传输层协议。一些会话层的功能在一些传输层协议中已经存在。其次，应用层不只有一个软件，可以在这一层开发很多应用。如果一个特定的应用需要会话层和表示层的一些功能，那么可以在开发那个软件时包含进去。

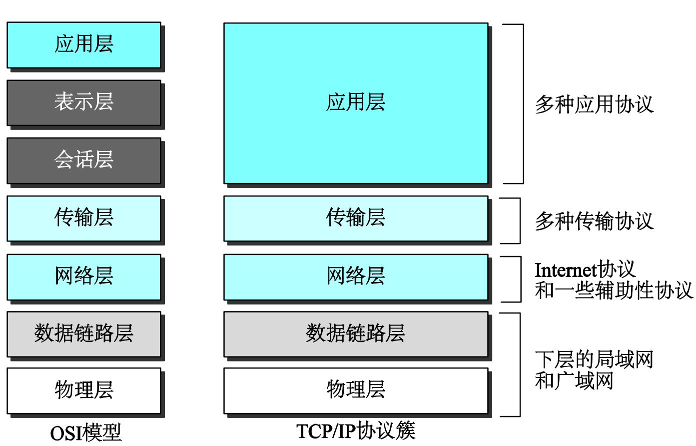


图 1-20　TCP/IP 与 OSI 模型

OSI 模型未成功的原因

OSI 模型在 TCP/IP 协议簇之后出现。多数专家起初都很振奋并认为 TCP/IP 协议将完全被 OSI

模型代替。这种事情没有发生有多种原因，我们只给出该领域所有专家都认同的 3 种原因。第一，OSI 在 TCP/IP 已经完全部署后才完成，很多时间和金钱已经花费在 TCP/IP 协议簇上；改变它将花费大量的资金。第二，OSI 模型中的一些层次从来没有完整定义过。例如，尽管表示层和会话层提供的服务列于文档中，但是这两层的实际协议既没有完整地定义也没有完整地描述，相应的软件也没有完整地开发出来。第三，当一个组织在不同的应用中实现 OSI 时，OSI 没有表现出足够高的性能，以诱使 Internet 管理机构从 TCP/IP 协议簇转向 OSI 模型。

1.3 Internet 发展史

我们已经给出了 Internet 及其协议的概况，现在简单介绍 Internet 的发展史。这个简单的历史可以使我们看到在不到 40 年的时间里，Internet 怎样从一个私有网络演变成一个全球的网络。

1.3.1 早期历史

1960 年之前存在一些通信网络，如电报网络和电话网络。这些网络适合于那个时代的恒定速率通信，即两个用户之间的连接建立之后，就可以传输编码的信息（电报）或声音（电话）。另一方面，计算机网络应该能够处理突发的数据，这意味着网络应能够在不同的时间按照不同的速率接收数据。整个世界正在等待分组交换网络的出现。

分组交换网络的诞生

1961 年，MIT 的 Leonard Kleinrock 首次提出了针对突发流量的分组交换网络。同一时期，兰德研究院（Rand Institute）的 Paul Baran 和英国国家物理实验室（National Physical Laboratory in England）的 Donald Davies 也发表了一些关于分组交换网络的文章。

ARPANET

20 世纪 60 年代中期，研究机构中的大型机是独立的设备。不同厂家生产的计算机不能相互通信。美国国防部（DOD）高级研究计划署（Advanced Research Projects Agency，ARPA）对寻找连接计算机的方法产生了兴趣，其目的是使他们资助的研究者能够共享他们的发现，从而减少投资和降低重复劳动。

在 1967 年美国计算机协会（Association for Computing Machinery，ACM）的一次会议上，ARPA 提出了组建一个连接计算机的小型网络——高级研究计划署网络（Advanced Research Projects Agency Network，ARPANET）的想法。按照这种想法，每台计算机（不必是同一生产厂家的计算机）都将附加一台称为接口信息处理机（interface message processor，IMP）的特定计算机。反过来，IMP 将被相互连接在一起。每个 IMP 不但能够与其他的 IMP 通信，而且能够与它自己连接的主机通信。

1969 年，ARPANET 变成了现实。位于加利福尼亚大学洛杉矶分校（University of California at Los Angeles，UCLA）、加利福尼亚大学圣巴巴拉校区（University of California at Santa Barbara，UCSB）、斯坦福研究院（Stanford Research Institute，SRI）和犹他大学（University of Utah）的 4 个结点通过 IMP 连接在一起，形成了一个网络。一种称为网络控制协议（Network Control Protocol，NCP）的软件提供主机之间的通信。

1.3.2 Internet 的诞生

1972 年，ARPANET 项目组的核心成员 Vint Cerf 和 Bob Kahn 开始合作开展所谓的网络互联项目（Internetting Project）。他们希望连接不相同的网络使得一个网络上的主机能够与另一个网络上的主机进行通信。需要克服的问题有很多：不同的分组大小、不同的接口类型、不同的传输速率，以及不同的可靠性需求。Cerf 和 Kahn 提出利用被称为网关（gateway）的一种设备作为中间硬件，进行一个网络到另一个网络的数据传输。

TCP/IP

Cerf 和 Kahn 在 1973 年里程碑式的文章中描绘了实现端到端数据投递的协议。这是一个新版本的 NCP。这篇关于传输控制协议（TCP）的文章包括了封装、数据报、网关的功能等概念。其主要思想是把纠错功能从 IMP 移到了主机。ARPA 的这个 Internet 现在变成了通信领域关注的焦点。大约在这个时期，ARPANET 转交由防御通信署（Defense Communication Agency，DCA）负责。

1977 年 10 月，一个包含了 3 种不同网络（ARPANET、分组无线电网络、分组卫星网络）的互联网成功地展示在人们面前。现在，网络之间的通信变成了可能。

之后不久，官方决定将 TCP 分成两个协议：传输控制协议（Transmission Control Protocol，TCP）和因特网协议（Internet Protocol，IP）。IP 处理数据报的路由选择，TCP 负责高层次的功能，如分段、重组、检错。这个新的联合体变成了人们知道的 TCP/IP。

1981 年，按照与国防部的协议，UC 伯克利（UC Berkeley）将 UNIX 操作系统进行修改使其包含了 TCP/IP。包含有网络软件的流行操作系统对网络的普及起了很大的作用。伯克利 UNIX 的开放性（非厂商相关的）实现使厂商能够获得工作代码，并基于这些代码构建他们的产品。

1983 年，官方废除了初始的 ARPANET 协议，TCP/IP 变成了 ARPANET 的正式协议。如果人们希望使用 Internet 访问处于不同网络的计算机，那么他们需要运行 TCP/IP 协议。

MILNET

1983 年，ARPANET 分裂成两个网络：军队用户使用的**军用网络**（Military Network，MILNET）和非军队用户使用的 ARPANET。

CSNET

Internet 历史上的另一个里程碑是 1981 年创建的 CSNET。**计算机科学网**（Computer Science Network，CSNET）是由美国**国家科学基金委**（National Science Foundation，NSF）资助的一个网络。该网络是为由于缺乏与国防部的联系而不能加入 ARPANET 的大学设计的。CSNET 是一个造价不高的网络，它没有冗余的链路并且传输速率也较低。

20 世纪 80 年代中期，美国大部分具有计算机科学系的大学成为了 CSNET 的一部分。其他一些研究机构和公司也构建了他们自己的网络并利用 TCP/IP 进行互联。起初，Internet 一词与政府资助构建的连接网络相关，现在指利用 TCP/IP 协议连接的网络。

NSFNET

随着 CSNET 的成功，NSF 在 1986 年开始资助**国家科学基金网络**（National Science Foundation Network，NSFNET）。NSFNET 是一个连接美国 5 个超级计算机中心的主干网。由于这个 1.544Mbps 的 T-1 主干网允许社区网络接入，因此可以在全美范围内提供网络连接。1990 年，ARPANET 正式退出历史舞台并被 NSFNET 代替。1995 年 NSFNET 又恢复到它科学研究网络的初始理念。

ANSNET

1991 年，美国政府认为 NSFNET 已不能支撑迅速增长的 Internet 流量。IBM、Merit 和 Verizon 三个公司填充了这段真空，组建了一个称为先进网络与服务（Advanced Network & Services，ANS）的非盈利组织，搭建了一个新的、高速 Internet 主干网。该主干网被称为**高级网络服务网**（Advanced Network Services Network，ANSNET）。

1.3.3 今天的 Internet

今天，我们见证了基础设施和新应用的迅速增长。今天的 Internet 是向全世界提供服务的信息港。使 Internet 变得如此流行的是新应用的发明和出现。

万维网

20 世纪 90 年代看到的 Internet 应用的急速增长得益于万维网（World Wide Web，WWW）的

出现。Web 是由欧洲原子核研究委员会(CERN)的 Tim Berners-Lee 发明的。这个发明增加了 Internet 的商业性应用。

多媒体

多媒体应用最近的发展，如 IP 语音（电话）、IP 视频（Skype）、视频共享（YouTube）和 IP 电视（PPLive）的出现，增加了网络用户的数量和用户在网时间。我们将在第 8 章讨论多媒体。

对等应用

对等（peer-to-peer，P2P）网络也是一个新的、具有很大潜力的通信领域。我们将在第 2 章介绍一些 P2P 应用。

1.4　标准和管理

在对 Internet 和它的协议的讨论中，我们经常看到一些参考标准或管理实体。在本节，我们为不熟悉这些标准和管理实体的读者介绍这些标准和管理实体；熟悉这些内容的读者可以跳过本节。

1.4.1　Internet 标准

Internet 标准是一个彻底通过测试的规范，该规范对从事互联网工作的人员非常有用。Internet 标准是一个必须遵循的正式的规则。经过严格的过程，一个规范才能达到 Internet 标准的某一状态。规范开始于 Internet 草案。一个 **Internet 草案**（Internet draft）是一个工作文档（该项工作正在进行中），没有官方的状态，具有 6 个月的生命周期。在 Internet 管理机构建议下，草案可以作为**请求评论**（Request for Comment，RFC）文档发布。每个 RFC 都经过编辑并被分配一个序号，所有感兴趣的团体均可获得。RFC 历经多个成熟阶段，并按照它们要求的级别分类到不同的类别。

成熟阶段

在一个 RFC 生命周期中，它会处于 6 个成熟阶段（maturity levels）之一：建议标准（proposed standard）、草案标准（draft standard）、Internet 标准（Internet standard）、历史的（historic）、实验性的（experimental）和信息性的（informational），如图 1-21 所示。

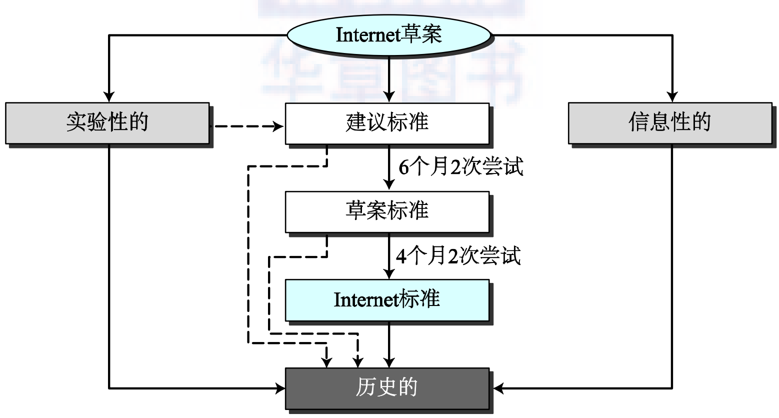


图 1-21　RFC 的成熟阶段

- **建议标准**。建议标准是一个稳定的、得到很好理解的、Internet 社会对其有足够兴趣的规范。在这个阶段，该规范通常被多个不同的工作组进行测试和实现。
- **草案标准**。在至少两个独立的和协作的实现之后，建议标准升级到草案标准。除非遇到困难，一个草案标准在遇到特定的问题进行修改之后，通常会变成 Internet 标准。
- Internet **标准**。在成功的实现证实之后，一个草案标准就达到了 Internet 标准状态。
- **历史的**。从历史的观点看，历史的 RFC 非常重要。它们或者被后来的规范所取代，或者从

未通过必要的成熟阶段而变成 Internet 标准。

- **实验性的**。归类为实验性的 RFC 描述了与一种实验环境相关的工作，该实验环境不影响 Internet 的运行。这样一个 RFC 不应该在任何实用的 Internet 服务中实现。
- **信息性的**。归类为信息性的 RFC 包含了与 Internet 相关的通用的、历史的或指导性的信息。这类文章通常由非 Internet 组织的人员编写（如厂商）。

要求的级别

RFC 分为 5 个*要求的级别*（requirement level）：要求的（required）、推荐的（recommended）、可选的（elective）、限制使用的（limited use）和不推荐的（not recommended）。

- **要求的**。如果一个 RFC 必须被所有 Internet 系统实现，以获得最小的一致性，那么该 RFC 被标识为"*要求的*"。例如，IP（见第 4 章）和 ICMP（见第 4 章）都是要求的协议。
- **推荐的**。标识为"推荐的"RFC 不是为了获得最小一致性而必须要求的；推荐它是因为它有用。例如，FTP（见第 2 章）和 TELNET（见第 2 章）都是推荐的协议。
- **可选的**。标识为"可选的"RFC 既不要求必须实现也不是推荐的。但是为了自身的利益，系统可以使用它。
- **限制使用的**。标识为"限制使用的"RFC 仅仅应该在限制的环境下使用。多数实验性的 RFC 归类为这个级别。
- **不推荐的**。标识为"不推荐的"RFC 对通常的应用是不合适的。通常历史性的（遭到反对的）RFC 可能归类到这个级别。

RFC 文档可以在网站 http://www.rfc-editor.org 获得。

1.4.2　Internet 管理

起初以研究为主的 Internet 逐渐演化并赢得了以重要商业活动为主的大量用户。协调 Internet 问题的不同工作组引导了这种增长和发展。附录 D 给出了一些工作组的地址、e-mail 地址和电话号码。图 1-22 显示了 Internet 的管理组织。

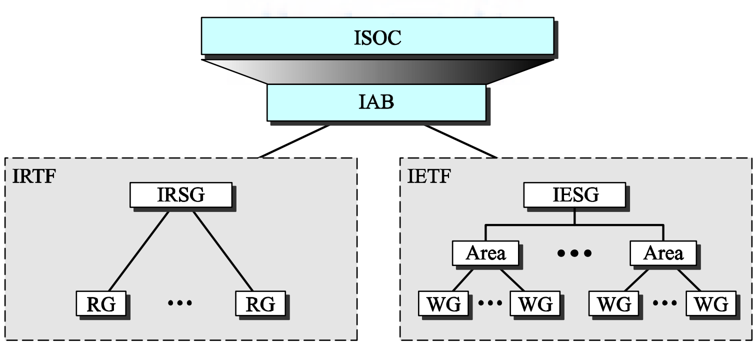


图 1-22　Internet 的管理

ISOC

成立于 1992 年的 **Internet 协会**（Internet Society，ISOC）是一个国际性、非盈利的组织，主要提供 Internet 标准处理过程的支持。该支持主要通过维护和支持其他 Internet 管理小组实现，如 IAB、IETF、IRTF 和 IANA（见下面的介绍）。ISOC 也负责推进与 Internet 相关的研究与其他学术活动。

IAB

Internet 体系结构委员会（Internet Architecture Board，IAB）是 ISOC 的技术顾问。IAB 的主要目标是关注 TCP/IP 协议簇的持续发展，作为技术顾问向 Internet 社会的研究成员提供服务。IAB

通过它的 Internet 工程任务组（Internet Engineering Task Force，IETF）和 Internet 研究任务组（Internet Research Task Force，IRTF）两个主要部门实现这个目标。IAB 的另一个责任是编辑和管理前面描述的 RFC 文档。IAB 也负责 Internet 与其他标准化组织和论坛的联系。

IETF

Internet 工程任务组（Internet Engineering Task Force，IETF）是一个由 Internet 工程指导组（Internet Engineering Steering Group，IESG）管理的一个工作组论坛。IETF 负责确认运行中的问题并提出解决方案。IETF 也开发和审查计划作为 Internet 标准的规范。工作组分为几个领域，每个领域集中精力于一个特定的内容。目前划分了 9 个领域，这些领域包括了应用、协议、路由、下一代网络（IPng）管理和安全。

IRTF

Internet 研究任务组（Internet Research Task Force，IRTF）是一个由 Internet 工程指导组（Internet Engineering Steering Group，IESG）管理的一个工作组论坛。IRTF 关注于与 Internet 协议、应用、结构和技术相关的、长期的研究课题。

IANA 和 ICANN

直到 1998 年 10 月，美国政府支持的 **Internet 号码分配管理局**（Internet Assigned Numbers Authority，IANA）负责 Internet 域名和地址的管理。之后，由一个国际委员会管理的非营利机构——**Internet 名称与编号分配组织**（Internet Corporation for Assigned Names and Numbers，ICANN）——承担了 IANA 的工作。

网络信息中心（NIC）

网络信息中心（Network Information Center，NIC）负责收集和发布有关 TCP/IP 的信息。

Internet 组织的地址和网站列于附录 D 中。

1.5　章末资料

推荐读物

有关本章更详细的内容讨论，我们推荐参阅下列书籍、网站和 RFC 文档。括号中的条目为本书末尾参考文献中的索引号。

书与文章

包括[Seg 98]、[Lei et al. 98]、[Kle 04]、[Cer 89]和[Jen et al. 86]在内的一些书和文章完整地覆盖了 Internet 的历史。

RFC 文档

RFC 791（IP）和 RFC 817（TCP）专门讨论了 TCP/IP 协议簇。在后面的章节中，我们将列出在每层中与每个协议相关的不同 RFC 文档。

小结

网络是一个由通信链路连接起来的设备集合。设备可以是计算机、打印机或其他具有发送和（或）接收网络中另一结点所产生的数据的设备。今天我们谈到的网络主要分为两种类型：局域网和广域网。目前，Internet 由很多广域网和局域网通过连接设备和交换站点连接而成。大部分希望连接 Internet 的最终用户需要利用 ISP 提供的服务。ISP 分为主干 ISP、区域 ISP 和本地 ISP。

协议是管理通信的规则集。在协议分层中，我们需要遵循两个原则以提供双向通信。首先，每一层需要实现两个相反的任务。其次，位于两端每层下的两个对象应该是等同的。TCP/IP 是一个由 5 个层次组成的层次化协议，这 5 层为应用层、传输层、网络层、数据链路层和物理层。

互联网的历史开始于 20 世纪 60 年代中期的 ARPA 网。Internet 的诞生与 Cerf 和 Kahn 的工作，以及连接网络的网关出现有很大关系。Internet 的管理随着 Internet 的发展不断演化。ISOC 促进和发起了相关的研究和活动。IAB 是 ISOC 的技术顾问组。IETF 是负责运行问题的工作组论坛。IRTF 为关注于长期发展研究课题的工作组论坛。ICANN 负责 Internet 域名和地址的管理。NIC 负责收集和发布有关 TCP/IP 协议的信息。

Internet 标准是通过完全测试的规范。Internet 草案为非官方的工作文档，具有 6 个月的生命周期。一个草案可能被作为 RFC 文档发布。RFC 经过成熟阶段并按照要求级别分成不同的类别。

1.6　习题集

测试题

本章的交互式测试题请参见本书的网站。在进行其他练习之前，强烈建议学生完成这些测试题以检查对这些内容的理解程度。

练习题

Q1-1　局域网中利用一条公共电缆进行传输（如图 1-1a 所示）是不是一种广播（一对多的）传输？请解释。

Q1-2　在一个具有链路层交换机的局域网中（如图 1-1b 所示），主机 1 希望向主机 3 发送消息。由于通信需要通过链路层交换机，这个交换机需要拥有一个地址吗？请解释。

Q1-3　如果每个局域网都要能够与其他局域网直接通信，那么连接 n 个局域网需要多少个点到点的广域网？

Q1-4　当我们使用本地电话与朋友通话时，我们使用的是电路交换网还是分组交换网？

Q1-5　当一个家庭用户利用拨号或 DLS 服务连接 Internet 时，电话公司承担什么角色？

Q1-6　为了进行双向通信，我们在这一章讨论的协议分层需要遵循的第一个原则是什么？

Q1-7　链路层交换机包含 TCP/IP 协议簇的哪些层？

Q1-8　一个路由器连接 3 条链路（网络）。这个路由器包含以下列出的哪些层？

a．物理层　　**b**．数据链路层　　**c**．网络层

Q1-9　在 TCP/IP 协议簇中，当我们思考应用层的逻辑连接时，发送方和接收方的对等体是什么？

Q1-10　一台主机利用 TCP/IP 协议簇与另一台主机通信。在下面列出的层次中发送和接收的数据单元分别是什么？

a．应用层　　**b**．网络层　　**c**．数据链路层

Q1-11　下列哪些数据单元被封装在帧中？

a．用户数据报　　**b**．数据报　　**c**．段

Q1-12　下列哪些数据单元是从用户数据报中解封装出来的？

a．数据报　　**b**．段　　**c**．消息

Q1-13　下列哪些数据单元具有应用层的消息和第 4 层的头部？

a．帧　　**b**．用户数据报　　**c**．比特

Q1-14　列出本章提到的一些应用层协议。

Q1-15　如果一个端口号为 16 位（2 个字节），那么 TCP/IP 协议簇中传输层的最小头部大小是多少？

Q1-16　下面列出的层次使用的地址（标识符）类型分别是什么？

a．应用层　　**b**．网络层　　**c**．数据链路层

Q1-17　当我们说传输层多路复用和多路分解应用层消息时，我们的意思是不是说传输层能够把应用层的多个消息合并到一个数据分组中？请解释。

Q1-18　你能解释我们为什么没有提到应用层的多路复用/多路分解吗？

Q1-19　假设我们要把两台相互隔离的主机连接在一起，以使它们能够相互通信。在两台主机之间需要链路层交换机吗？

Q1-20　如果源主机和目的主机之间有一条单独的通道，在两台主机之间需要路由器吗？

Q1-21　解释 Internet 草案和建议标准的不同。

华章图书

Q1-22　解释要求的 RFC 和推荐的 RFC 之间的不同。

Q1-23　解释 IETF 和 IRTF 承担的任务有什么不同。

思考题

P1-1　在图 1-10 中，当 Maria 向 Ann 发送信息时，回答下列问题：

a．在 Maria 一端，第 1 层向第 2 层提供的服务是什么？

b．在 Ann 一端，第 1 层向第 2 层提供的服务是什么？

P1-2　在图 1-10 中，当 Maria 向 Ann 发送信息时，回答下列问题：

a．在 Maria 一端，第 2 层向第 3 层提供的服务是什么？

b．在 Ann 一端，第 2 层向第 3 层提供的服务是什么？

P1-3　假设 2010 年连接 Internet 的主机数为 5 亿台，如果主机数按照每年 20%的速率增长，那么 2020 年的主机数是多少？

P1-4　假设一个系统采用 5 个协议层次，如果应用程序构建了一个 100 字节的消息，每一个层次（包括第 5 层和第 1 层）向数据单元增加 10 字节的头部，那么这个系统的效率（应用层字节数与传输字节数的比值）是多少？

P1-5　假设我们构建了一个分组交换互联网。我们需要利用 TCP/IP 协议簇传输一个巨大的文件。发送大分组的优势和劣势各是什么？

P1-6　将下列语句匹配到 TCP/IP 协议簇的一层或多层：

a．路由判定

b．连接到传输介质

c．为最终用户提供服务

P1-7　将下列语句匹配到 TCP/IP 协议簇的一层或多层：

a．构建用户数据报

b．负责处理相邻结点间的帧

c．把比特变换为电磁信号

P1-8　在图 1-18 中，当 IP 协议解封传输层分组时，它怎么知道这个分组应该投递到哪个上层协议（UDP 或 TCP）？

P1-9　假设一个私有互联网在数据链路层采用 3 个不同的协议（L1、L2 和 L3）。按照这种假设，重画图 1-18。是否可以这样说，在数据链路层我们在源结点进行多路分解，在目的结点进行多路复用？

P1-10　假设一个私有互联网要求加密/解密应用层消息，以保证其安全。如果我们需要增加一些关于加密/解密处理的信息（例如进行处理的算法），那么是不是意味着我们向 TCP/IP 协议簇添加了一层？如果你认为是这样，重画 TCP/IP 层次（图 1-12 的 b）。

P1-11　协议分层可以在我们生活的很多地方找到。想象你到度假胜地进行一个双程旅行。起飞前，你需要在你本地的机场进行一些处理工作。当你到达度假胜地机场后，你也需要进行一些处理工作。使用行李托运/行李提取、登机/下机、起飞/着陆等层次，对双程旅行进行协议分层。

P1-12　在图 1-4 中，从西海岸的一台主机到东海岸的一台主机仅有一条单一的通路。在这个互联网中，我们为什么需要两台路由器？

P1-13　在今天的 Internet 中，数据表示变得越来越重要。一些人主张 TCP/IP 协议簇应该增加一个新的层次以负责数据表示（参见附录 C）。如果将来增加这个新的层次，那么它应该处于协议簇的什么位置？重画图 1-12 以包含这个层次。

P1-14　在一个互联网中，我们用新的局域网技术替换原有的局域网技术。在 TCP/IP 协议簇中，哪些层次需要改变？

P1-15　假设一个应用层协议利用 UDP 提供的服务进行编写，那么这个应用层协议不改变就可以使用 TCP 服务吗？

P1-16　利用图 1-4 显示的互联网，描述当西海岸的一台主机与东海岸的一台主机交换信息时，TCP/IP 协议簇的层次和数据流。

1.7　模拟实验

Applets

一种查看实际的网络协议和观察一些示例解决方案的方法是利用交互式的动画。我们构建了一些 Java 小程序用于展示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些协议。

实验作业

进行网络和网络设备相关的实验至少可以采用两种方法。在第一种方法中，我们可以构建一个隔离的网络实验室，利用网络硬件和软件模拟在每章讨论的内容。我们可以构建一个互联网，从一台主机发送信息到另一台主机，从而观察分组流并测量其性能。尽管这种方案比第二种方法更有效、更适宜于教学，但是这种方案实现起来比较昂贵，不是所有的单位都能够投资这样一个独立的实验室。

在第二种方法中，我们可以使用 Internet——世界上最大的网络，作为我们的虚拟实验室。我们可以利用 Internet 发送和接收分组。一些免费的、可以下载的软件允许我们捕获和观察交换的数据分组。我们可以分析这些分组以理解网络的理论概念是如何付诸实现的。尽管第二种方法不能控制和改变分组的路由以观察 Internet 的行为，效果不如第一种方法明显，但是这种方法实现起来比较廉价。它不需要实际的实验室，利用我们的桌面机或笔记本就可以实现。同时，需要的软件也可以免费下载。

很多 Windows 和 UNIX 操作系统下的程序和工具允许我们嗅探、捕获、跟踪和分析我们的电脑和 Internet 之间交换的数据分组。一些程序和工具，如 WireShark 和 Ping Plotter，具有图形用户接口（GUI）；其他如 traceroute、nslookup、dig、ipconfig 和 ifconfig 为命令行式的网络管理工具。这些工具对网络管理员调试网络、学生学习网络都非常有价值。

在本书中，尽管我们偶尔采用其他工具，但是大部分实验作业使用的是 Wireshark。Wireshark 从一个网络接口捕获活动的分组数据并对详细的协议信息进行显示。但是，Wireshark 是一种被动的分析器。它仅仅能“计量”网络上的东西但不能操纵它们；它既不能在网络上发送数据分组也不能做其他主动的操作。另外，Wireshark 也不是一种入侵检测工具。它不会对网络入侵发出警告。但是，它能帮助网络管理员或者网络安全工程师理解网络内部的状况，帮助他们解决网络故障。Wireshark 除了对网络管理员和安全工程师必不可少之外，它对协议开发人员也非常有价值，协议开发人员可以使用 Wireshark 对实现的协议进行调试。与此同时，Wireshark 也是一个很好的教学工具，学习计算机网络的学生能够使用它实时地观察协议操作的细节。

在本节实验作业中，我们学习如何下载和安装 Wireshark。下载和安装指南在本书网站的第 1 章实验部分给出。在这个文档中，我们还讨论了该软件背后的基本想法、它的窗口格式，以及怎样使用它。这个实验的学习可以使学生为今后完成采用 Wireshark 的实验作业做好准备。

第 2 章

Computer Networks: A Top-Down Approach

应　用　层

整个因特网、硬件以及软件的设计和开发就是为应用层提供服务。TCP/IP 协议簇的第五层正是这些服务的所在位置。其他四层协议使这些服务成为可能。学习因特网技术的一种方法就是先解释应用层提供的服务，然后再展示其他四层是如何支持这些服务的。由于本书正是依照这种方法，因此应用层是我们首先要讨论的内容。

在因特网的发展历程中，创造和使用了许多应用协议。有些是有特定用途但从未成为标准的。有些已经被弃用。有些被修改或者被新的协议所替换。一些协议得以幸存下来并成为标准应用。新的应用协议被持续不断地加入因特网中。

本章分 5 节讨论应用层。

- 2.1 节将介绍因特网提供的服务的本质以及两个应用类型：传统类型即客户-服务器模式（client-server paradigm），以及新类型即对等模式（peer-to-peer paradigm）。
- 2.2 节讨论客户-服务器模式的概念以及这个模式是如何为因特网用户提供服务的。
- 2.3 节讨论一些客户-服务器模式的预定义和标准应用。我们也会讨论一些流行的应用，比如万维网、文件传输、电子邮件等等。
- 2.4 节讨论对等模式中的概念及协议。我们会介绍一些协议，诸如 Chord、Pastry 和 Kademilia。我们也会提及一些使用这些协议的流行应用。
- 2.5 节我们给出在客户-服务器模式下如何通过用 C 语言编写两个程序创建一个新的应用。这两个程序一个是为客户端编写的，另一个是为服务器编写的。在第 11 章我们将展示如何用 Java 语言编写客户-服务器程序。

2.1　介绍

应用层为用户提供服务。通信是由逻辑连接提供的，这意味着两个应用层假设存在一个假想的直接连接，通过这个连接可以发送和接收报文。图 2-1 展示了这种逻辑连接背后的思想。

图 2-1 展示了这样一幅场景，一个科学家在名为天空研究所的研究公司工作，她需要从网络书商那里订购一本与自己研究相关的书。一条逻辑连接在天空研究所电脑的应用层与科技著作服务器的应用层之间建立了。我们把第一台主机叫 Alice，把第二台主机叫 Bob。应用层的通信是逻辑的，而不是物理的。Alice 和 Bob 认为他们之间有一条双向逻辑信道，他们可以通过这条信道发送接收报文。然而，实际的通信通过了若干设备（Alice、R2、R4、R5、R7 以及 Bob）以及图上所示的若干物理信道。

2.1.1　提供服务

在因特网出现之前就开始运营的所有通信网络都被设计成向网络用户提供服务。然而，这些网络大都原先被设计为提供一种特定服务。比如电话网络原先被设计为提供语音服务，它允许全世界的人们相互交谈。然而之后，这个网络用于了其他服务，比如传真，用户在两个终端加上一些额外的硬件可以收发传真了。

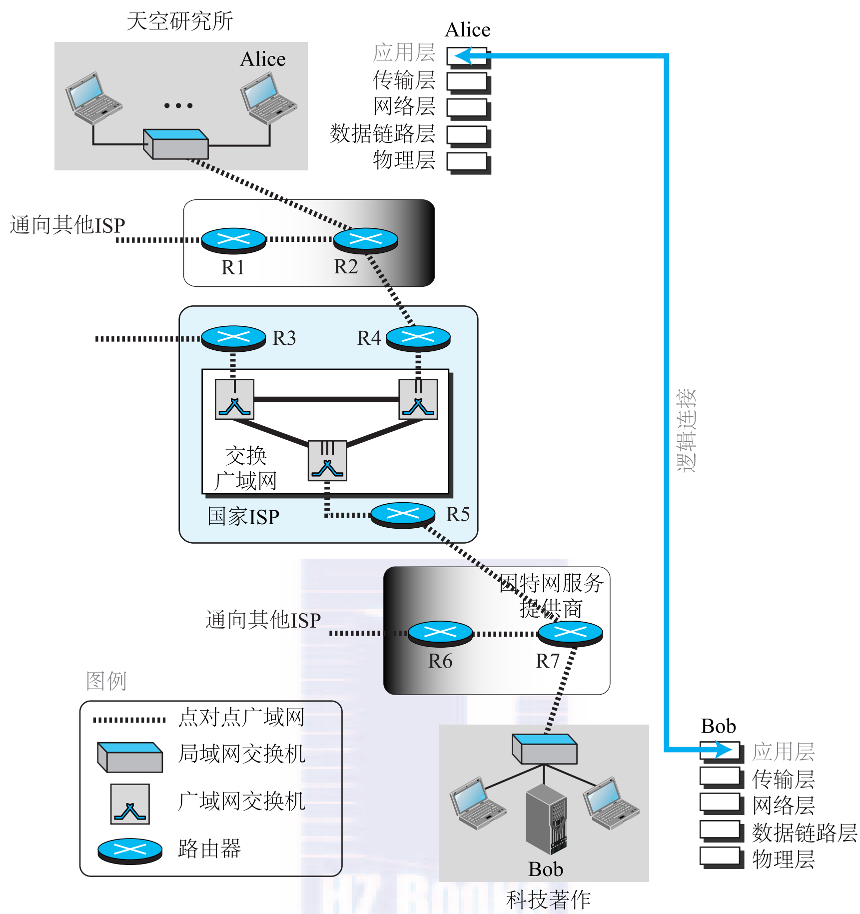


图 2-1　应用层逻辑连接

因特网原先是为同一个目的而设计出来的，即为全世界的用户提供服务。然而 TCP/IP 协议簇的层次结构使得因特网比其他网络更加灵活，诸如邮件网络和电话网络。协议簇的每一层原先由一个或多个协议组成，但是可以加入新的协议，因特网管理机构可以删除或替换某些协议。然而如果将一个协议增加到一层中，那么这个协议应当被设计成使用底层协议提供的服务。如果从一层中去除一个协议，那么应该注意去改写它的上一层协议，高层协议可能使用了它提供的服务。

然而由于应用层是协议簇的最高层，它与其他层有些不同。这层中的协议不为任何其他协议提供服务，它们只接收来自传输层协议的服务。这意味着，可以从这层中轻易地去除协议。只要新的协议可以使用传输层协议提供的服务，那么就可以把它加入到这一层。

由于应用层是唯一向因特网用户提供服务的层次，因此如上所述，应用层的灵活性允许新的应用协议轻松地加入因特网，这一点在因特网的发展历程中不断发生。当因特网被创建时，只有很少的应用协议可以供用户使用。但是现在，我们无法给出这些协议的数目，因为新的协议正在源源不断地被添加进去。

标准和非标准协议

为了使因特网流畅运作，需要标准化和归档 TCP/IP 协议簇前四层所使用的协议。它们通常成为包的一部分，包含在如 Windows 或 UNIX 的操作系统里。然而为了灵活，应用层协议既可以标准化也可以非标准化。

标准应用层协议

有一些应用层协议已经被因特网管理机构标准化和归档，并且我们与因特网的日常交流中正在使用它们。每个标准协议是一对程序，它们与用户和传输层进行交互，传输层为用户提供特定的服务。在本章稍后我们会讨论一些标准应用，某些会在其他章节讨论。至于这些应用协议，我们需要知道它们提供什么服务类型，它们如何工作，以及这些应用给我们哪些选项等等。学习这些协议可以使网络管理员更容易解决使用过程中的问题。对这些协议工作方式的深入理解也将会使我们了解如何创建新的非标准协议。

非标准应用层协议

如果一个程序员能编写两个程序，那么她就可以创建非标准应用层程序，这两个程序通过与传输层交互为用户提供服务。本章的后面我们将给出如何编写这样的程序。如果私人使用的话，创建一个非标准（专利的）协议甚至不需要因特网管理机构的批准，这使得因特网在世界上十分流行。一个私人公司可以创建一种新的定制应用协议，来和遍布全球的办公室进行通信，公司使用 TCP/IP 协议簇前四层提供的服务而不使用任何一个标准应用程序。所需要的就是以一种计算机语言来编写程序，这些程序使用传输层协议提供的服务。

2.1.2　应用层模式

应该弄清楚的是，为了使用因特网，我们需要两个应用程序彼此交互：一个运行在世界某个地方的电脑上，另一个运行在世界其他地方的另一台电脑上。两个程序需要通过因特网基础设施彼此发送报文。然而，我们还没有讨论这两个程序之间的关系。两者都应该能够请求和提供服务吗？抑或应用程序仅仅实现这两种功能中的一个？为了回答这个问题，在因特网的发展历程中开发了两种模式：客户-服务器模式和对等模式。此处，我们简要介绍这两种模式，但是我们会在稍后讨论它们的细节。

传统模式：客户-服务器

传统模式称为**客户-服务器模式**。在几年前它还是最流行的。在这种模式中，服务提供者是一个称为服务进程的应用程序，它不断地运行着，等待另一个称为客户进程的应用程序通过因特网建立连接并请求服务。通常有一些服务进程可以提供特定类型的服务，但是有很多客户向这些服务进程请求服务。服务进程必须一直运行，当需要接受服务时客户进程就被打开。

客户-服务器模式与某些因特网领域外的服务类似。比如，任何地方的电话号码查询中心都可以被看做服务器，一个打电话询问特定电话号码的用户可以被看做客户。电话号码查询中心必须每时每刻准备提供服务，当需要服务时用户可以给中心致电一小段时间。

尽管客户-服务器模式的通信是在两个应用程序之间的，但是每个程序的角色是全然不同的。换言之，我们不能把一个客户端程序当做服务器程序运行，反之亦然。在本章的后面，当谈论在这种模式下的客户-服务器编程时，我们总是要分别为这两种服务类型编写应用程序。图 2-2 展示了一个客户-服务器通信的例子，其中三个客户与一个服务器进行通信，客户接受服务器提供的服务。

这个模式的问题是通信负荷集中在服务器上，这意味着服务器应该是一台强大的计算机。但是，即使是强大的计算机也会难以应对大量客户同时尝试连接。另一个问题是应该存在一个服务提供商，它乐于接受这项花费并创建一台提供特定服务的强大服务器，这意味着服务必须为服务器产生收益，以此便可促进这种安排。

很多传统该服务仍然在使用这种模式，包括万维网（World Wide Web，WWW）以及它的传播媒介：超文本传输协议（HyperText Transfer Protocol，HTTP）、文件传输协议（File Transfer Protocol，FTP）、安全人机界面（Secure Shell，SSH）、电子邮件等等。我们在本章后面讨论这些协议和应用。

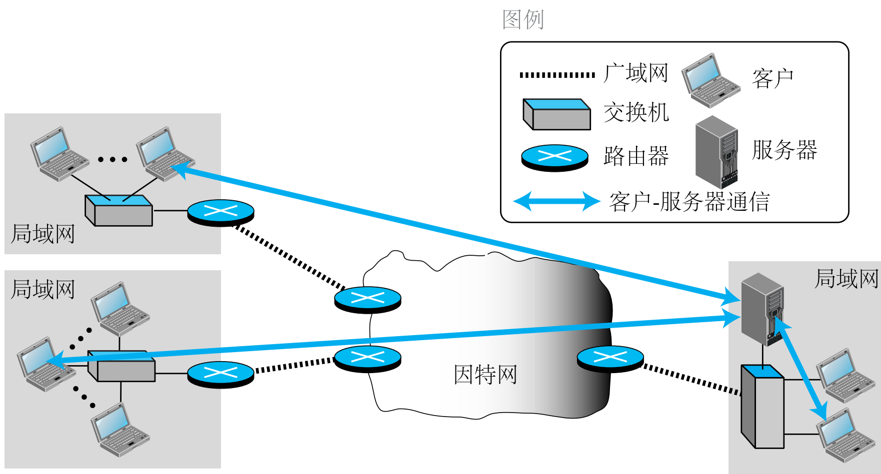


图 2-2　客户-服务器模式示例

新模式：对等

一个称为**对等模式**（通常简称为 P2P 模式）的新模式已经出现，它迎合了新应用的需求。在这种模式下，不需要一个不断运行且等待客户进程连接的服务器进程。责任在对等结点（peer）之间分担。连接到因特网的计算机可以在这一次提供服务却在下一次接受服务。一台计算机甚至可以同时接受和提供服务。图 2-3 展示了这种模式的通信例子。

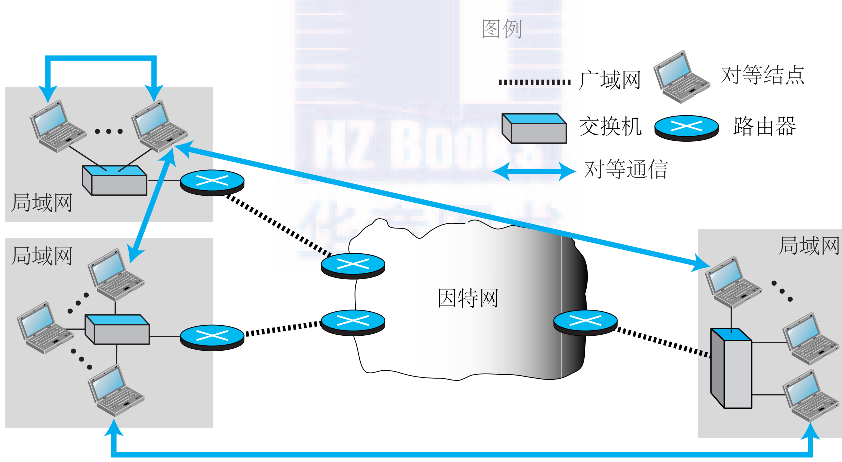


图 2-3　对等模式示例

一个真正符合这个模式的领域是网络电话。电话通信确实是对等活动，没有一方需要不断运行来等待另一方呼叫。当某些计算机有东西要彼此共享而连到因特网上时，也会使用到对等模式。比如，如果某因特网用户有一个可以和其他用户共享的文件，那么这个用户没有必要去建立服务器并一直运行服务器进程等待其他用户连接并获取文件。

由于对等模型无需一直运行和维护昂贵的服务器，它是容易扩展且经济划算的。尽管如此，还是存在一些挑战的。主要挑战就是安全问题，在分布式服务之间创建安全通信比在那些由专用服务器控制的服务之间建立安全通信要更困难。另一个挑战就是适用性，似乎并不是所有的应用都可以使用这个新模式。比如倘若某一天网络可以作为对等服务执行，并不会有非常多的因特网用户准备参与进来。

有一些新的应用使用这种模式，诸如 BitTorrent、Skype、IPTV 以及网络电话。我们将在后面

讨论其中的一些应用，对另一些应用的讨论将放到后续的章节。

混合模式

一个应用可以通过结合这两种模式的优点来把这二者混合起来。比如轻量级的客户–服务器通信可以用来寻找可以提供服务的对等结点（peer）的地址。当找到这个地址时，实际服务可以通过使用对等模式从对等结点中获得。

2.2　客户–服务器模式

在客户–服务器模式中，应用层的通信是在两个运行着的应用程序之间进行的，这两个应用程序称为**进程**（process）：客户和服务器。客户是一个运行着的程序，它通过发送请求初始化通信；另一个应用程序是服务器，它等待来自客户的请求。服务器处理来自客户的请求，准备结果并将其发送给客户。服务器的定义意味着当一个来自客户的请求到达时，服务器必须是正在运行的，但是客户不必这样，它只在必要的时候运行。这意味着如果我们有两台相互连接的电脑，我们可以在一台电脑上运行客户进程，在另一台上运行服务器进程。然而，我们需要小心的是服务器进程要在客户端程序运行前开启。换言之，服务器的生存期是无限的：它应该开启后一直运行，等待客户。客户的生存期是有限的：它通常发送有限的请求给对应的服务器，接收响应然后停止。

2.2.1　应用程序接口

客户进程是如何与服务器进程进行通信的？一个计算机程序通常是由预定义了指令集的计算机语言编写的，这个指令集告诉计算机要做什么。计算机语言有一个数学操作指令集、一个字符串处理指令集、一个输入/输出访问指令集等。如果我们需要一个进程与另一个进程通信，那么我们就需要一个新的指令集告知 TCP/IP 协议簇的低四层打开连接，发送数据，从另一个终端接收数据，以及关闭连接。这样的指令集通常称为**应用程序接口**（Application Programming Interface，API）。程序中的接口是两个实体之间的指令集。在这种情况下，一个实体是应用层中的进程，另一个是操作系统，操作系统封装了 TCP/IP 协议簇的前四层。换句话说，电脑制造商将协议簇的前四层编写进操作系统中并包含了 API。这样，当通过因特网发送和接收分组时，应用层运行的进程才能够与操作系统通信。有许多通信 API 被设计出来。其中三个是很常见的：**套接字接口**、**传输层接口**（Transport Layer Interface，TLI）以及 STREAM。在这一节里，我们仅简要讨论最常见的套接字接口，以给出一个应用层网络通信的宏观概念。

在 20 世纪 80 年代，套接字接口作为 UNIX 环境的一部分出现在加州伯克利大学。如图 2-4 所示，套接字接口是提供应用层和操作系统间通信的指令集，是一个可以被某进程用来与另一个进程进行通信的指令集。

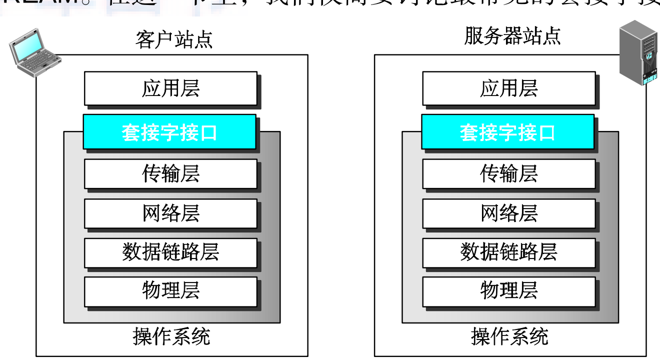


图 2-4　套接字接口的位置

套接字的概念允许我们使用编程语言中对其他信源和信宿设计的所有指令的集合。比如，在绝大多数计算机语言中，如 C、C++以及 Java，已经有很多可以从信源读数据和向信宿写数据的指令。这些信源和信宿有：键盘（信源）、监视器（信宿）以及文件（信源和信宿）。我们可以使用相同的指令从套接字里读或向套接字里写入。换言之，我们只不过在向编程语言中加入新的信源和信宿，而没有改变发送和接收数据的方式。图 2-5 展示了这种思想并将套接字与信源和信宿进行比较。

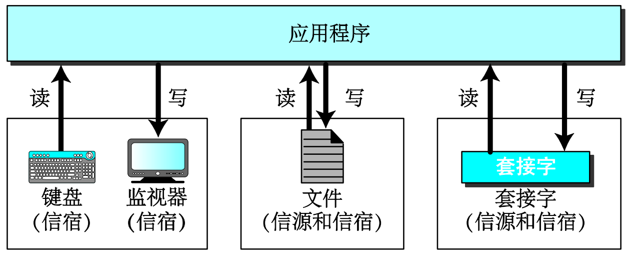


图 2-5 套接字和其他信源和信宿采用相同的方式

套接字

尽管套接字在行为上应该和一个终端或文件类似，但是它不是物理实体，而是一种抽象。套接字是供应用程序创建和使用的数据结构。

我们可以说，就应用层而言，客户进程和服务器进程间的通信是两个套接字间的通信。如图 2-6 所示，在两个终端创建了两者间的通信。客户认为套接字是接收请求和发出响应的实体；服务器认为套接字是发出请求并且需要获得响应的实体。如果我们创建两个套接字，一端创建一个，并且正确定义源端和目的端地址，那么我们就可以使用指令去发送和接收数据了。其余就是操作系统以及嵌入的 TCP/IP 协议的工作了。

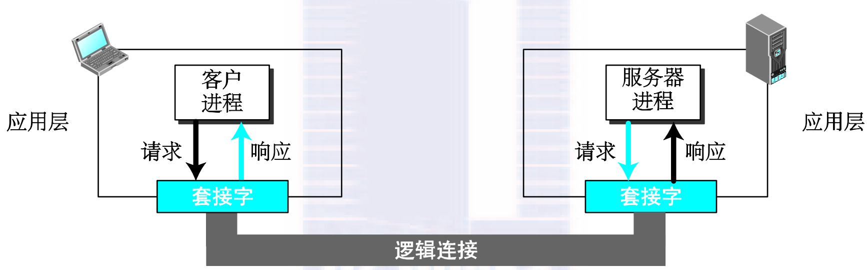


图 2-6 对等通信中套接字的使用

套接字地址

客户和服务器的交互是双向通信。在双向通信中，我们需要一对地址：本地地址（发送端）和远程地址（接收端）。在一个方向上的本地地址对另一个方向来说就是远程地址，反之亦然。由于客户–服务器模式的通信是在套接字之间的，我们需要一对**套接字地址**（socket address）：一个本地套接字地址和一个远程套接字地址。然而，我们需要以 TCP/IP 协议簇的标识符来定义套接字地址。

一个套接字地址首先定义了一个客户或服务器所在的计算机。正如我们将在第 4 章讨论的，因特网上的一台计算机由 IP 地址唯一确定，IP 地址在现在的因特网版本中是一个 32 位的整数。然而，可能在同一时间同一台计算机上有很多客户或服务器进程运行，这意味着我们需要另一个标识符来定义特定的通信中所涉及的客户或服务器。正如我们将在第 3 章讨论的，一个应用程序可以由端口号定义，它是一个 16 位整数。这意味着套接字地址应该是一个 IP 地址和一个端口号的组合，如图 2-7 所示。

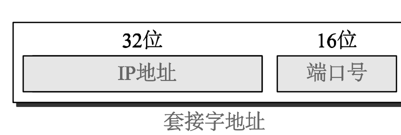


图 2-7 套接字地址

由于套接字定义了通信终端，我们可以说套接字是由一对套接字地址标识的，这一对套接字地址分别是本地套接字地址和远程套接字地址。

例 2.1 在电话通信中我们可以找到两级地址。一个电话号码可以定义一个组织，电话分机号码可以定义组织内的一个特定连接。这样，电话号码就像 IP 地址一样定义了整个组织；分机号码就像端口号，它定义了特定的连接。

寻找套接字地址

客户或服务器如何寻找一对套接字地址来通信呢？每个站点的情况是不同的。

服务器站点

服务器需要一个本地（服务器）和一个远程（客户）套接字地址来通信。

本地套接字地址 本地（服务器）套接字地址由操作系统提供。操作系统知道运行着服务器进程的计算机的 IP 地址。然而服务器进程的端口号需要被分配。如果这个服务器进程是因特网管理结构定义的标准进程，那么端口号就已经分配好了。比如，超文本传输协议（HTTP）被分配的端口号是 80，其他进程就不能再使用了。我们将在第 3 章讨论这些熟知的端口号。如果服务器进程不是标准进程，那么它的设计者就要在规定范围内选择一个端口号，并分配给进程。当服务器开始运行时，它就得知了本地套接字地址。

远程套接字地址 对服务器来说，远程套接字地址是建立连接的客户套接字地址。由于服务器可以给多个用户提供服务，它事先并不知道远程套接字地址。当客户试图连接服务器时，服务器可以知道这个套接字地址。客户套接字地址包含在发送给服务器的请求报文中，它成为远程套接字地址来给客户提供响应。换言之，尽管服务器的本地套接字地址是固定的并且在生存期内一直使用，但是远程套接字地址在服务器与不同客户进行交互时都会改变。

客户站点

客户也需要一个本地（客户）和一个远程（服务器）套接字地址来通信。

本地套接字地址 本地（客户）套接字地址也由操作系统提供。操作系统知道运行着客户进程的计算机的 IP 地址。然而端口号是每次客户进程需要开始通信时分配给客户进程的一个临时 16 位整数。但是端口号需要从一组由因特网管理机构定义的整数中分配，这称为临时端口号，我们将在第 3 章深入探讨。操作系统需要确保新的端口号没有被其他正在运行的客户进程所占用。

远程套接字地址 然而找到远程（服务器）套接字地址需要更多的工作。当一个客户进程开启时，它应该知道自己想要连接到的服务器的套接字地址。这里有两种情况。

- 有时，开启客户进程的用户知道运行着服务器进程的计算机的端口号和 IP 地址。这通常在我们编写客户和服务器应用并进行测试时发生。比如在本章的结尾，我们将编写一个简单的客户–服务器程序，并且我们将采用此种方式进行测试。在这种情况下，当运行客户端程序时程序员可以提供这两条信息。
- 尽管每个标准应用都有一个熟知端口号，但绝大多数情况下我们不知道 IP 地址。这会在如下情景下发生：连接网页、给朋友发送电子邮件以及从一个远程站点拷贝文件等。在这些情况下，服务器有一个名称，一个唯一标识服务器进程的标识符。例如 URL 就是这种标识符，像是 www.xxx.yy 或者电子邮件地址 xxxx@yyyy.com。客户进程现在需要将这个标识符（名称）改成对应的服务器套接字地址。由于端口号应该是一个熟知端口号，因此客户进程通常知道端口号。IP 地址可以通过使用另外一个客户–服务器应用来获得，这个应用叫做域名系统（Domain Name System，DNS）。稍后我们会在本章讨论 DNS，但是我们知道它工作起来就像因特网中的电话簿就够了。将此情景与电话簿相比较。我们想给某个已知姓名的人打电话，但是那个人的电话号码可以从电话簿上得到。电话簿将姓名映射到电话号码；DNS 将服务器名称映射到运行着那个服务器的计算机 IP 地址上。

2.2.2 使用传输层的服务

一对进程向因特网中的用户提供服务，这些用户可以是人，也可以是程序。但是由于应用层没有物理通信，这一对进程需要使用传输层提供的服务来通信。正如我们在第 1 章简要讨论的，在 TCP/IP 协议簇中有三个常见的传输层协议：UDP、TCP 以及 SCTP，这些会在第 3 章详细讨论。绝

大多数标准应用被设计来使用这些协议。当我们编写一个新的应用时我们可以决定使用哪个协议。对于传输层协议的选择将严重影响应用进程的性能。在这一节，我们首先讨论每个协议提供的服务，来帮助大家理解为什么一个标准应用会使用它，以及编写一个新应用时需要使用哪个协议。

UDP 协议

UDP 提供了无连接的、不可靠的数据包服务。无连接服务意味着两个交换报文的终端之间没有逻辑连接。每个报文都是独立的实体，它被封装在一个称为**数据报**（datagram）的分组中。UDP 看不到来自同一个源端并去往同一个目的端的数据报之间的关系（连接）。

UDP 是不可靠的协议。尽管它可能在传输中检查数据是否被破坏，但是它并不要求发送端重传被破坏的或丢失的数据。对于某些应用，UDP 有一个优势，即它是面向报文的。它保留报文边界。

我们可以将无连接、不可靠的服务与邮局提供的常规服务进行对比。两个实体可以在它们之间交换信件，但是邮局并没有看见这些信件之间的任何连接。对于邮局，每个信件都是带有它自己的发送者和接收者的独立实体。尽管邮局是尽力而为的，但是如果一个邮件在发送过程中丢失或被损坏，邮局概不负责。

如果应用程序发送小报文，并且简单性和速度要比可靠性更重要，那么可以将这个应用程序设计成使用 UDP 协议的程序。比如，某些管理和多媒体应用符合这个分类。

TCP 协议

TCP 提供面向连接的可靠的字节流传输。TCP 要求两个终端首先通过交换一些连接建立分组（connection-establishment packet）来建立一个逻辑连接。这个阶段有时称为握手，它设定了两个终端间的某些参数。这些参数包括要交换的数据分组大小、用于保存数据直到整个报文全部到达的缓冲区的大小等等。在握手过程后，两个终端可以向着彼此的方向以报文段形式发送数据块。通过计算交换的字节数，可以检测字节的连续性。比如，如果某些字节丢失或损坏了，接收端可以请求重发这些字节，这使得 TCP 成为一个可靠协议。我们将在第 3 章看到 TCP 也可以提供流量控制和拥塞控制。TCP 协议的一个问题是它不是面向报文的，它不保留报文边界。

我们可以将 TCP 提供的面向连接的可靠的服务与电话公司的服务进行比较，尽管这种比较仅仅是在某种程度上进行的。如果两方决定通过电话而不是邮局通信，他们可以创建一次连接，进行一段时间的通话。电话服务在某种程度上是可靠的，因为如果一个人没听明白或听不清另一方所说的话，他可以要求对方再说一遍。

绝大多数需要发送长报文以及要求可靠性的标准应用都从 TCP 的服务中受益。

SCTP 协议

SCTP 提供了前面两个协议组合的功能。就像 TCP 一样，SCTP 提供了面向连接的可靠的服务，但是它不是面向字节流。它是像 UDP 一样面向报文的。除此之外，SCTP 可以通过提供多媒体网络层连接提供多媒体流服务。

SCTP 通常适用于那些不但需要可靠性，而且即使网络层连接发生错误也需保持连接不断开的应用。

2.3　标准客户–服务器应用

在因特网的发展历程中，很多客户–服务器应用被开发出来。我们没有必要重定义它们，但是我们需要理解它们做了什么。对于每一个应用，我们也需要知道有哪些可用选项。学习这些应用以及了解它们提供不同服务的方式可以帮助我们将来创建定制的应用。

我们在这一节选择了六个标准应用。我们从 HTTP 和万维网开始，因为几乎所有的网络用户都使用它们。然后，我们介绍文件传输和电子邮件应用，这些在因特网上占很大的流量。接下来，我们解释远程登录以及如何通过 TELNET 和 SSH 协议实现远程登录。最后，我们讨论 DNS，所有应

用程序都使用它来将应用层标识符映射到相应的主机 IP 地址上。

其他章节将会适当讨论一些其他的应用，如动态主机配置协议（DHCP）和简单网络管理协议（SNMP）。

2.3.1　万维网和 HTTP

在这一节，我们首先介绍万维网（简称 WWW 或 Web）。之后我们讨论超文本传输协议（HTTP），它是与 Web 相关的最常见的客户−服务器应用程序。

万维网

Web 的思想最早由 Tim Berners-Lee 在 1989 年于 CERN㊀提出的，CERN 是欧洲原子研究中心（European Organization for Nuclear Research）的简称。它允许多个研究者在欧洲的不同地点访问彼此的研究。商业化的 Web 出现在 20 世纪 90 年代早期。

今天的 Web 是信息宝库，其中称为网页的文档在全世界分布，并且相关的文档链接在一起。Web 的流行和成长与前面介绍过的两个术语有关：分布式的（distributed）和链接的（linked）。分布式允许 Web 增长。世界上每个 Web 服务器都可以增加一个新的网页到这个宝库中并向所有因特网用户宣告，而这不会使一些服务器超载。链接使得一个网页与另一个存储在世界某个地方的主机上的网页相互引用。网页的链接通过使用一个称为超文本（hypertext）的概念而实现，这个概念在因特网出现之前很多年就被引入进来了。这个思想是当一个链接出现在文档中时，利用一台机器自动获取存储在系统中的另一个文档。Web 用电子方式实现了这个思想：当用户点击这个链接时允许获取被链接的文档。现在超文本这个术语的含义已经由一开始的被链接的文本文档变成了超媒体（hypermedia），这表示网页可以是文本文档、图片、音频文件或视频文件。

Web 的用途已经超越了简单获取被链接的文件。现在，它用于提供电子购物和游戏。一个用户可以在任何时候使用网络来收听广播节目或收看电视节目，而不必只有在这些节目播出的时候才能收听或收看。

结构

如今 WWW 是一个分布式客户−服务器服务。使用浏览器的用户可以访问一个正在服务器上运行的服务。然而服务是分布在很多称为站点（site）的地点上。每一个站点有一个或多个文档，它们称为网页。但是，每个**网页**（web page）可以包含一些到其他网页的链接，那些被链接的网页可以在同一个站点也可以在其他站点。换言之，一个网页可以是简单的也可以是复合的。简单网页没有链接；复合网页有一个或多个链接。每个网页是一个有名字和地址的文件。

例 2.2　假设我们需要获取一个科学文档，这个文档包含了到另一个文本文件和一幅大图片的引用。图 2-8 说明这个场景。

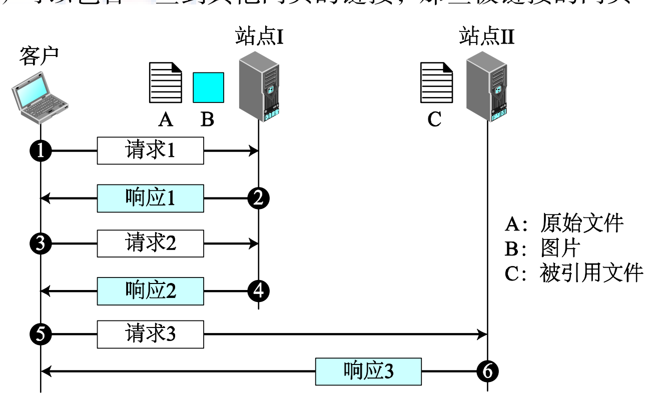


图 2-8　例 2.2

主文档和图片存储在同一个站点的两个不同文件中（文件 A 和文件 B）；被引用的文本文件被存储在另一个站点（文件 C）。由于我们正在处理三个不同的文件，如果想看到全部文档就需要三项事务。第一项事务（请求/响应）获取主文档的一份拷贝（文件 A），这个文

㊀ 法语：Conseil European pour la Recherche Nuclear（欧洲核子研究理事会）。

件有对于第二个和第三个文件的引用（指针）。当获得和浏览主文档的拷贝时，用户可以单击图片的引用来引起第二项事务并获取图片（文件 B）的拷贝。如果用户需要看到被引用的文本文件的内容，她可以单击这个链接（指针）引起第三项事务并获取文件 C 的拷贝。注意，尽管文件 A 和文件 B 都存储在站点 A 上，但是它们是有不同名称和地址的独立文件。需要两项事务才能获取它们。非常重要的一点是我们需要记住，例 2.2 中的文件 A，B 和 C 是独立的网页，每一个都有独立的名称和地址。对于文件 B 或 C 的引用尽管包含在文件 A 中，但这并不意味着不能单独地获取每一个文件。第二个用户可以用一项事务来获取文件 B。第三个用户可以用一项事务获取文件 C。

网络客户（浏览器） 很多供应商提供可以解释和显示网页的商业浏览器（browser），它们都使用几乎相同的结构。每个浏览器通常包含三部分：控制程序、客户协议和解释程序（见图 2-9）。

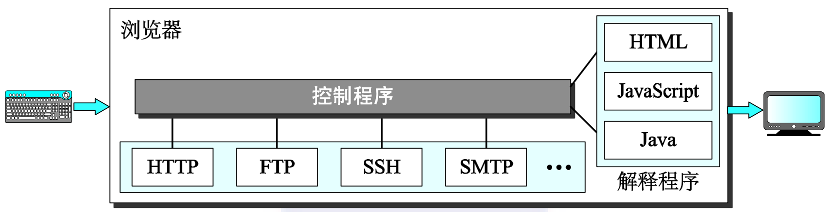


图 2-9 浏览器

控制程序接收来自键盘和鼠标的输入并使用客户端程序来访问文档。在文档被访问后，控制程序使用解释程序来在屏幕上显示文档。客户协议可以是后文描述的任何一种协议，比如 HTTP 或 FTP。解释程序可以是 HTML、Java 或 JavaScript，这要取决于文档的类型。商业浏览器包括：Internet Explorer、Netscape Navigator 以及 Firefox。

网络服务器 网页是存储在服务器上的。每当请求到达时，相应的文档就被发送到客户。为了提高效率，服务器通常将被请求的文件存储在内存的缓存中；访问内存比访问磁盘快。服务器也可以通过多线程或多进程来提高效率。某些流行的网页服务器包含了 Apache 和微软互联网信息服务器。

统一资源定位符（Uniform Resource Locator，URL）

作为文件，网页需要有一个唯一的标识符来与其他网页区别开。为了定义一个网页，我们需要三个标识符，主机、端口以及路径。然而在定义网页之前，我们需要告知浏览器我们想使用哪个客户-服务器应用，这称为协议。这意味着我们需要四个标识符来定义网页。第一个是用来获取网页的运载工具形式，后三项组合在一起定义了目的对象（网页）。

- **协议**。第一个标识符是我们使用的用来访问网页的客户-服务器程序的缩写。尽管绝大多数情况下的协议是 HTTP（超文本传输协议），这个协议我们将简要讨论，但是我们也会使用其他协议，比如 FTP（文件传输协议）。
- **主机**。主机标识符可以是服务器的 IP 地址或主机被给予的唯一名称。IP 地址可以按点分隔的十进制数表示法定义（如 64.23.56.17），第 4 章对其进行了描述；名称通常是唯一定义主机的域名，比如 forouzan.com，我们将在这一章后面的域名系统（DNS）讨论。
- **端口**。端口是一个 16 位整数，通常为客户-服务器应用而预定义。例如，如果用 HTTP 协议来访问网页，那么熟知端口号就是 80。然而，如果使用一个不同的端口号，那么这个号码将被显示给出。
- **路径**。路径定义了下层的操作系统中文件的位置和名称。这个标识符的格式通常依赖于操作系统。在 UNIX 中，路径是后面接有文件名的一组目录名，它们通过斜杠来分隔。比如，/top/next/last/myfile 是一个唯一定义 myfile 文件的路径，这个文件存储在目录 last 中，last 目录是 next 目录的一部分，而 next 又在目录 top 下。换言之，路径自顶向下列出目录，后接文件名。

为了把这四部分组合在一起，便设计出了**统一资源定位符**（Uniform Resource Locator，URL）；它在四个部分之间用三个不同的分隔符，如下所示：

protocol://host/path	绝大多数情况使用
protocol://host:port/path	当需要标识出端口号时使用

例 2.3　URL **http://www.mhhe.com/compsci/forouzan/**定义了与本书作者相关的一个网页。字符串 **www.mhhe.com** 是 McGraw-Hill 公司的计算机名称（**www** 这三个字母是主机名的一部分，它被加到商业主机的前面）。路径是 **compsci/forouzan/**，它定义了在目录 **compsci**（计算机科学）下 Forouzan 的网页。

网上文档

万维网的文档可以分为三大类：静态文档、动态文档和活动文档。

静态文档（Static Document）　静态文档是在服务器中创建和存储的固定内容的文档。客户只能得到一个文档的副本。换言之，文件的内容在创建文件时就决定了，而不是使用时决定的。当然，服务器中的内容是可以改变的，但是用户不能改变它们。当客户访问文档时，一个文档的副本被发送给用户。然后用户可以使用浏览器查看文档。静态文档使用如下语言编写：超文本标记语言（Hypertext Markup Language，HTML）、可扩展标记语言（Extensible Markup Language，XML），可扩展样式表语言（Extensible Style Language，XSL）以及可扩展超文本标记语言（Extensible Hypertext Markup Language，XHTML）。我们将在附录 C 中讨论这些语言。

动态文档（dynamic document）　当浏览器请求文档时动态文档就被网页服务器创建。当一个请求到达时，网页服务器运行一个应用程序或一个脚本来创建动态文档。服务器返回程序或脚本的结果作为对请求文档的浏览器的响应。由于对每个请求都要创建一个全新的文档，因此动态文档的内容随着请求的不同而不同。一个动态文档的简单例子是从服务器获取时间和日期。时间和日期是动态信息，因为它们不断变化。客户可能要求服务器去运行一个程序，比如 UNIX 中的 data 程序，然后发送程序结果给客户。尽管过去公共网关接口（Common Gateway Interface，CGI）用来获取动态文档，但是如今选择使用脚本语言如 Java Server Pages（JSP），JSP 是使用 Java 语言来编写脚本的，或者 Active Server Pages（ASP），这是使用 Visual Basic 语言编写脚本的微软产品，或者 ColdFusion，它将结构化查询语言（Structured Query Language，SQL）数据库中的查询嵌入到 HTML 文档中。

活动文档（active document）　对很多应用来说，我们需要在客户站点运行一个程序或脚本。这些称为**活动文档**。比如，假设我们想运行一个在屏幕上创建动画的程序或运行一个与用户交互的程序。程序当然需要在显示了动画或者发生了互动的客户站点运行。当浏览器请求一个活动文档时，服务器发送文档或脚本的一个副本。然后文档在客户站点（浏览器）那里运行。一种创建活动文档的方法是使用 Java applets，这是一种 Java 编写的位于服务器的程序。它被编译并准备运行。文档是字节码（二进制）形式的。另一种方法是使用 JavaScript，但是要在客户站点下载和运行这个脚本。

超文本传输协议（HTTP）

超文本传输协议（HyperText Transfer Protocol，HTTP）是一种用来定义客户服务器程序如何编写和如何从万维网获取网页的协议。一个 HTTP 客户发送一个请求；HTTP 服务器返回响应。服务器使用 80 端口号；客户使用一个临时端口号。HTTP 使用 TCP 服务，在之前讨论过，TCP 是一种面向连接的可靠的协议。这意味着，在客户和服务器进行任何事务之前，它们之间必须建立连接。在事务之后，连接应当终止。然而，客户和服务器不需要担心交换报文中的差错以及报文的丢失，因为 TCP 是可靠的而且将处理这个问题，我们将在第 3 章看到这一特性。

非持续与持续连接

正如我们在之前讨论的，嵌入到网页中的超文本概念可能需要多个请求和应答。如果网页，这个被获取的对象，位于不同的服务器，那么我们没有其他选择只能每获取一个对象就要创建一个新的 TCP 连接。然而，如果某些对象是位于同一台服务器的，我们可以有两种选择：一是每次使用

一个新的 TCP 连接获取一个对象，二是创建一个 TCP 连接获取全部对象。第一种方法称为非持续连接（nonpersistent connection），第二种称为持续连接（persistent connection）。在 HTTP1.1 版之前指定的是非持续连接，持续连接在 1.1 版中是默认的，但是可以被用户改变。

非持续连接　在非持续连接中，一个 TCP 连接被每一组请求/应答所创建。下面是这个策略的步骤：

1. 客户开启一个 TCP 连接并发送请求。
2. 服务器发送响应并关闭连接。
3. 客户读取数据直到它遇到了文件结束标记，然后关闭连接。

在这种策略中，如果文件包含了 N 个位于不同文件的图片连接（全都位于同一台服务器），那么必须开启和关闭连接 $N+1$ 次。非持续策略给服务器带来了高额开销，因为每次连接被开启时服务器都需要 $N+1$ 个不同的缓冲区。

例 2.4　图 2-10 展示了一个非持续连接的例子。客户需要访问一个包含图片链接的文件。文本文件和图片位于同一台服务器上。这里我们需要两个连接。对于每一个连接，TCP 需要至少三个握手报文来建立连接，但是请求可以和第三个报文一起发送。在连接建立之后，请求对象可以被发送。在接收到一个对象之后，需要另外三次握手报文来结束连接，我们将在第 3 章看到这一点。这意味着客户和服务器参与到这两次连接建立和连接终止之中。如果这项事务包含获取 10 个或 20 个对象，那么这些握手的往返时间总和将会是一个很大的开销。当在本章结尾展示客户–服务器编程时，我们将看到客户和服务器需要为每一个连接分配额外的资源，如缓冲区和变量。这是两个站点的另一个负担，而且对于服务器这边来说负担尤其重。

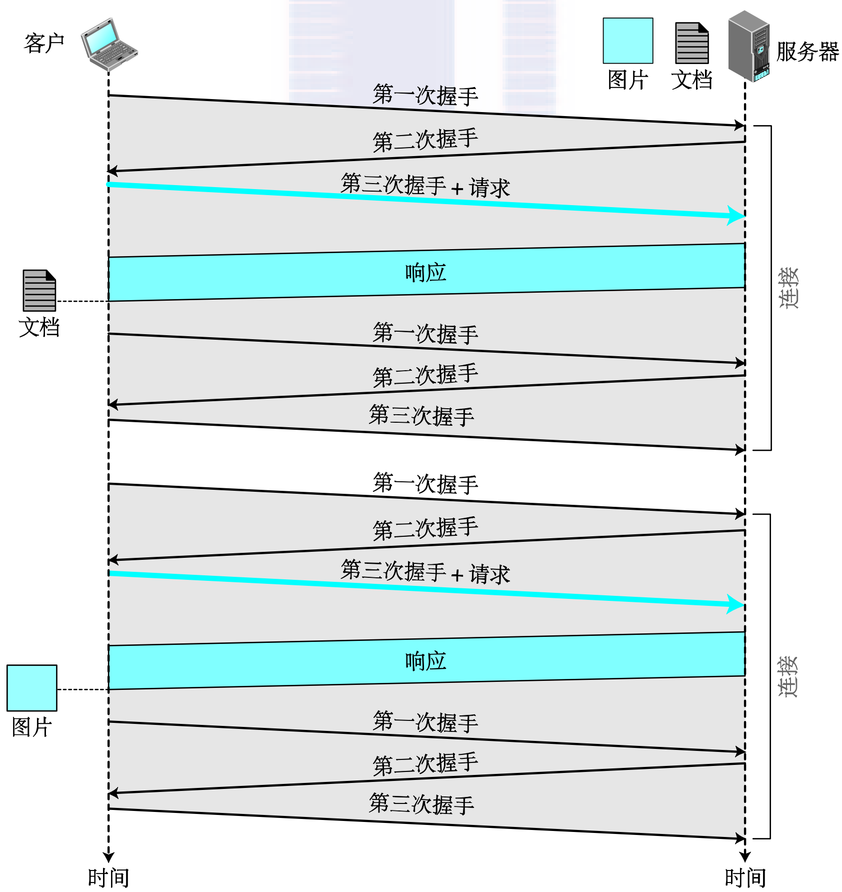


图 2-10　例 2.4

持续连接　HTTP1.1 版默认指定了持续连接。在持续连接中服务器在发送一个响应后，为响应更多的请求而将连接置为打开状态。服务器可以在客户的请求下或者在超时情况下将连接关闭。发送方通常在每次响应中发送数据长度。然而，偶尔情况下发送方不知道数据的长度。这是创建动态文档或活动文档时的情形。在这种情形下，服务器通知客户长度未知并在发送数据后关闭连接，因此客户知道数据已接收完毕。通过使用持续连接，可以节省时间和资源。每个站点只需要为连接设定一组缓冲区和变量。同时节省了连接建立和终止的往返时间。

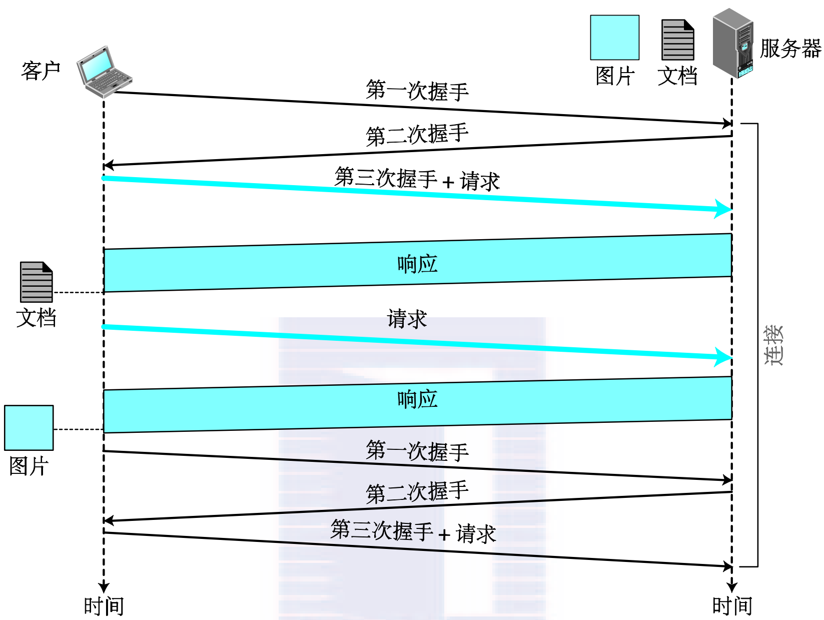


图 2-11　例 2.5

例 2.5　图 2-11 展示了与例 2.4 中相同的情景，但是使用的是持续连接。只有一个连接建立和终止，但是对于图片的请求是分别发送的。

报文格式

如图 2-12 所示，HTTP 协议定义了请求报文和响应报文的格式。我们把两种格式并列以示比较。每一种报文由四个部分组成。请求报文中的第一部分称为请求行；响应报文的第一部分称为状态行。其他三部分在请求报文和响应报文中有相同的名称。然而，这三部分只是名称相似，它们可能含有不同的内容。我们分别讨论每一个报文类型。

请求报文　正如之前所述，请求报文的第一行称为请求行。如图 2-12 所示，这一行有三部分由空格分隔开并且被两个字符（回车和换行）终止。这些字段称为方法、URL 和版本。

方法字段定义了请求类型。如表 2-1 所示，在 HTTP1.1 版中定义了若干种方法。绝大多数情况下，客户使用 GET 方法发送一个请求。在这种情况下，报文的主体是空的。当客户仅需要从服务器获得关于网页的信息，如上次修改的时间，这时使用 HEAD 方法。它也可以用来检测 URL 的有效性。这种情况下的响应报文只有头部；主体是空的。PUT 方法与 GET 方法是相反的；它允许客户将一个新的页面发送到服务器上（如果允许的话）。POST 方法与 PUT 方法类似，但是它用来发送一些信息到服务器上，这些信息被加入网页或用来修改网页。TRACE 方法用来调试；客户要求服务器回送请求来检查服务器是否正在获得请求。如果客户获得许可，DELETE 方法允许客户删除一个服务器上的网页。CONNECT 方法原先作为预留方法；后文会讨论到，这个方法可能被代

理服务器使用。最后，OPTIONS 方法允许客户询问网页属性。

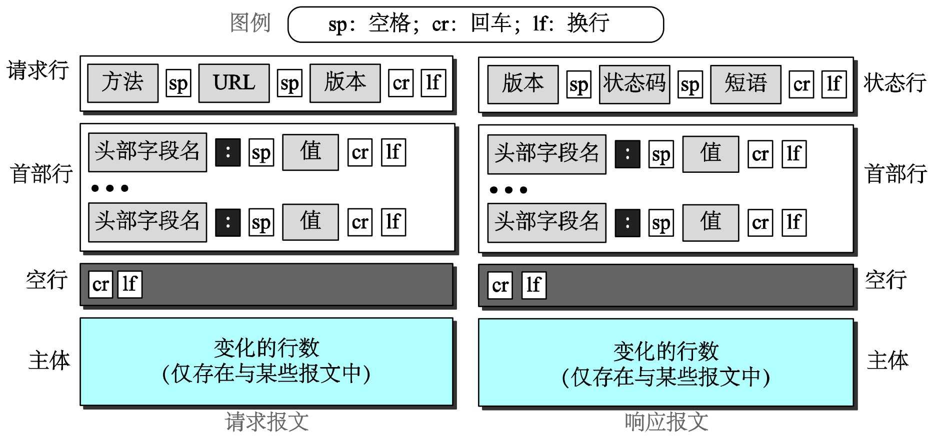


图 2-12　请求和响应报文格式

表 2-1　方法

方　法	动　作	方　法	动　作
GET	向服务器请求文档	TRACE	回送输入的请求
HEAD	请求关于文档的信息，而不是文档本身	DELETE	删除网页
PUT	从客户端向服务器发送文档	CONNECT	预留
POST	从客户端向服务器发送一些信息	OPTIONS	询问有关可用的选项

第二个字段：URL，在本章的前面部分已经讨论过了。它定义了相关网页的地址和名称。第三个字段：版本，给出了协议的版本，HTTP 最常用的版本是 1.1。

在请求行之后我们可以有一个或多个请求头部（request header）行。每一个头部行都从客户端向服务器发送额外的信息。例如，客户可以请求以某种特定格式发送文档。每个头部行有头部名字、一个冒号、一个空格和一个头部值（见图 2-12）。表 2-2 列出了一些请求中常用的头部名字。值字段定义了与每个头部名字相关的值。值列表可以在相应的 RFC 中查找到。

主体可以出现在请求报文中。通常，当使用 POST 或 PUT 方法时，它包含要发送的评论或要发布到网站上的文档。

表 2-2　请求头部名称

头　部	描　述	头　部	描　述
User-agent	标识客户端程序	Host	给出主机及客户端的端口号
Accept	给出客户端能够接受的媒体格式	Date	给出当前日期
Accept-charset	给出客户端可以处理的字符集	Upgrade	确定首选的通信协议
Accept-encoding	给出客户端可以处理的编码方案	Cookie	返回 cookie 给服务器（稍后解释）
Accept-language	给出客户端可以接受的语言	If-Modified-Since	如果文档在指定的日期之后被更新，则发送文档
Authorization	给出客户端有哪些许可		

响应报文　图 2-12 给出了响应报文的格式。响应报文包含状态行、头部行并且有时包含主体。响应报文的第一行称为状态行。这一行有三个字段，它们由空格分隔开并且被两个字符（回车和换行）终止。第一个字段定义了 HTTP 协议的版本，通常为 1.1。状态码字段定义了请求的状态。它包含三个数字。在 100 范围内的代码只代表一个报告，在 200 范围内的代码表示这是一个成功的请求。在 300 范围内的代码表示把客户端重定向到另一个 URL，在 400 范围内的代码表示在客户端发生错

误。最后，在 500 范围内的代码表示错误发生在服务器端。状态短语以文本格式解释了状态码。

在状态行之后，我们可以有一个或多个响应头部行。每一个头部行都从服务器向客户端发送额外的信息。例如，发送方可以发送关于文档的额外信息。每个头部行都有一个头部名称、一个冒号、一个空格和一个头部值。我们将在本节结尾列举一些头部行。表 2-3 列出了一些常用的头部名称。

表 2-3　响应头部名称

头　　部	描　　述	头　　部	描　　述
Date	给出当前日期	Content-Length	给出文档长度
Upgrade	确定首选的通信协议	Content-Type	指定媒体类型
Server	给出服务器信息	Location	指明新建或移动后文档的位置
Set-Cookie	服务器要求客户存储 cookie	Accept-Ranges	服务器将会接收的被请求的字节范围
Content-Encoding	指定编码方案	Last-modified	给出上次改变的日期和时间
Content-Language	指定语言		

主体包含了从服务器发送给客户的文档。除非响应是一个错误报文，否则主体是存在的。

例 2.6　这个例子获取了一个文档（见图 2-13）。我们使用 GET 方法来获取一个路径为 **/usr/bin/image1** 的图片。请求行给出了使用的方法（GET）、URL 以及 HTTP 版本（1.1）。头部有两行，它们表示客户可以接收 GIF 或 JPEG 格式的图片。请求是没有主体的。响应报文包含了状态行以及四个头部行。头部行定义了日期、服务器、内容编码（MIME 版本，在电子邮件部分将会描述）以及文档长度。文档主体在头部之后。

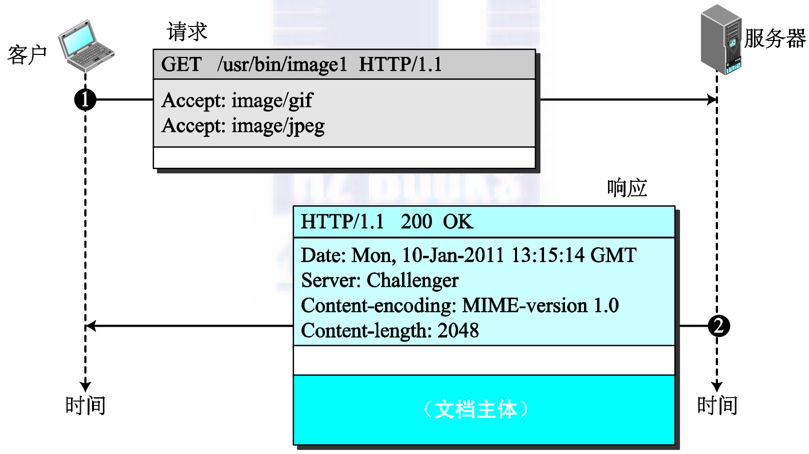


图 2-13　例 2.6

例 2.7　在这个例子中，客户要向服务器发送一个网页。我们使用 PUT 方法。请求行给出方法（PUT）、URL 以及 HTTP 版本（1.1）。其头部有四行。请求主体包含要发送的网页。响应报文包含状态行和四个头部行。被创建的文档是一个 CGI 文档，它包含在响应报文的主体中（见图 2-14）。

条件请求

客户可以在请求中加入条件。在这种情况下，如果条件满足，服务器将会发送被请求的网页或者通知用户。客户加入的最常见的一种条件是网页被修改的时间和日期。客户可以在发送请求时附带头部行 If-Modified-Since，这样来告知服务器客户只需要在指定日期之后更新的页面。

例 2.8　以下展示了一个客户在请求中加入了修改日期和时间的条件。

```
GET http://www.commonServer.com/information/file1 HTTP/1.1    请求行
If-Modified-Since: Thu, Sept 04 00:00:00 GMT                 头部行
                                                              空行
```

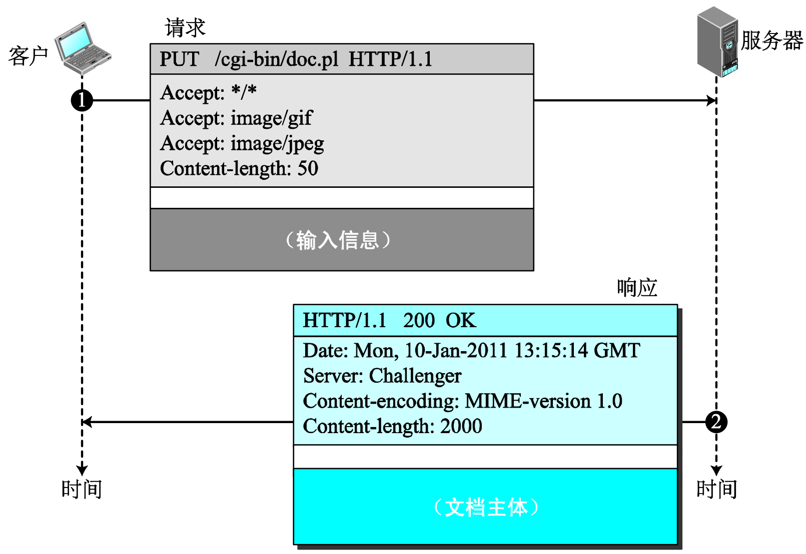


图 2-14 例 2.7

响应中的状态行表示在指定日期之后文档没有修改。响应报文的主体是空的。

HTTP/1.1 304 Not Modified	状态行
Date: Sat, Sept 06 08 16:22:46 GMT	头部第一行
Server: commonServer.com	头部第二行
	空行
(Empty Body)	空主体

Cookie

万维网起先被设计成无状态实体。客户发送请求；服务器响应。它们之间的关系就结束了。最初设计的 Web，公开检索可用的文档，完全符合这个目标。现在 Web 还有其他功能，如下所示：

- 网站作为电子商店（electronic store），允许客户在商店内浏览，选择需要的商品，把它们放入电子购物车内，最后使用信用卡付费。
- 有些网站只允许注册客户（registered client）访问。
- 有些网站是门户网站（portal）：用户可以选择他想看的网页。
- 有些网站仅作为广告（advertising）代理。

为了实现这些目的，cookie 机制就应运而生。

创建和存储 Cookie Cookie 的创建和存储与实现有关，然而它的原理是相同的。

1. 当服务器从客户端接收到请求后，它将客户端的信息存储在文件或字符串中。这些信息可能包含客户端的域名、cookie 内容（服务器收集到的关于客户端的信息，如主机名、注册号等）、时间戳，以及与实现有关的其他信息。

2. 服务器在响应中包含了它发送给客户端的 cookie。

3. 当客户端接收到响应后，浏览器在 cookie 目录中存储 cookie，并根据服务器域名进行分类。

使用 Cookie 当客户向服务器发送请求时，浏览器在 cookie 目录中查询是否有从那个服务器发送过来的 cookie。如果有，则在请求中包含这个 cookie。当服务器接收到这个请求后，它就知道了这是一个老客户，而不是新的。注意，cookie 的内容从来不让浏览器读取或者透露给用户，只由服务器创建并回收 cookie。现在让我们分析如何使用 cookie 来实现上面提到的四个功能。

- 网上电子商店（电子商务）可以为客户端的购物者使用 cookie。当客户端选择商品，并放入购物车中后，包含了这些商品信息（如它的数量、单价）的 cookie 就被发送到浏览器。

如果客户端选择第二个商品，cookie 就被新的选择信息所更新，依此类推。当客户端结束购物并准备付账离开时，就检索最终的 cookie，然后计算出总的费用。

- 当客户端第一次注册时，网站就向客户端发送一个 cookie，网站通过这种方式限制注册用户的访问。只有那些能够发送正确 cookie 的客户才能被允许今后重复访问。
- Web 门户以相同的方式使用 cookie。当用户选择她最喜爱的网页时，就生成一个 cookie 并发送到浏览器。当这个网站再次被访问时，这个 cookie 就发送给服务器说明这个客户端要查找什么页面。
- cookie 也用来作为广告代理。广告代理能够将大字标题广告放置在用户经常访问的网站的主页面上。广告代理仅提供指出大字标题广告地址的 URL，而不是大字标题广告本身。当用户访问网站主页并点击广告公司的图标时，一个请求就发送给了广告代理；广告代理就发送一个大字标题广告，如 GIF 文件，同时也包含了一个含有用户 ID 的 cookie。将来对这个大字标题广告的任何使用都会加入到一个分析用户 Web 行为的数据库中。广告代理已经收集了用户的爱好，并能够将这些信息卖给其他组织。这种 cookie 的使用方法引起很多争议，但愿今后能引入一些新的法规来保护用户的隐私信息。

例 2.9 图 2-15 展示了电子商店可以从 cookie 的使用中受益的场景。

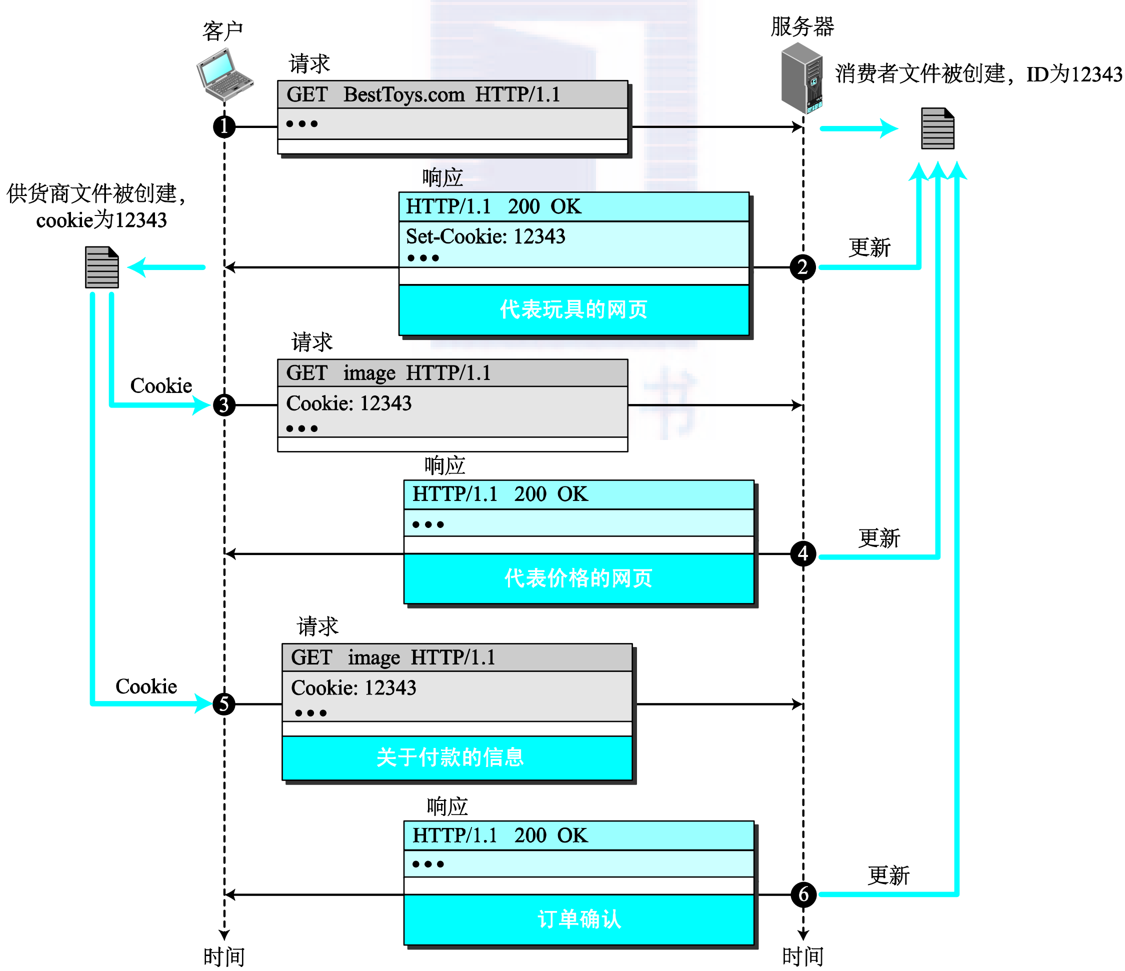


图 2-15 例 2.9

假设一个购物者想要从名叫 BestToys 的电子商店购买一个玩具。购物者浏览器（客户）发送一个请求到 BestToys 服务器。服务器为客户创建了一个空购物车（列表）并给购物车分配了一个 ID（如 12343）。服务器然后发送一个响应报文，它包含了所有可购买的玩具的图片，每个玩具下都有链接。如果单击，这个玩具就被选中。这个响应报文也包含了值为 12343 的 Set-Cookie 头部行。客户显示这些图片并且将 cookie 存储在名为 BestToys 的文件中。cookie 对购物者是不可见的。现在购物者选择一个玩具，单击它。客户发送了一个请求，但是 ID 号 12343 包含在 cookie 头部行。尽管服务器可能很繁忙并且忘记了消费者，但是当它接收到请求并检查头部时会发现 12343 这个值。服务器知道这个消费者不是新顾客；它搜寻带有 ID 12343 的购物车。购物车（清单）被打开，然后选中的玩具被插入到清单中。现在服务器发送另一个响应给购物者，告知她价钱并要求其支付款项。消费者提供她信用卡的信息并发送一个 cookie 值为 ID 12343 的新请求。当请求到达服务器时，它再次看到 ID 12343 并且接受订单以及付款，在响应中发送一个确认信息。关于客户的其他信息存储在服务器上。以后如果消费者访问这个商店，客户再次发送 cookie；商店会获取文档并且拥有关于用户的全部信息。

万维网高速缓存：代理服务器

HTTP 支持**代理服务器**（proxy server）。代理服务器是一台计算机，能够保存最近请求的响应的副本。HTTP 客户端向代理服务器发送请求。代理服务器检查本地高速缓存。如果高速缓存中不存在响应报文，代理服务器就向相应的服务器发送请求。返回的响应会发送到代理服务器中，并且进行存储，以用于其他客户端将来的请求。

代理服务器降低了原服务器的负载，减少了通信量并降低了延迟。但是，为了使用代理服务器，必须配置客户端访问代理服务器而不是目标服务器。

请注意，代理服务器既作为一个服务器又作为一个客户。当它收到客户的请求并有一个要发送给客户的响应时，它作为服务器并且发送响应给客户。当它收到客户的请求但没有要发送给客户的响应时，它首先作为客户然后发送请求给目标服务器。当响应被接受，它又作为服务器并发送响应给客户。

代理服务器位置

通常代理服务器位于客户站点。这意味着我们可以有如下代理服务器的层级：

1. 客户计算机也可以用作小容量代理服务器，它存储着与客户经常调用的请求相对应的响应。
2. 在一个公司，一个代理服务器可能安装在计算机 LAN 中来减少进出 LAN 的负载。
3. 带有很多客户的 ISP 可以安装一台代理服务器来减少进出 ISP 网络的负载。

例 2.10　图 2-16 给出了在诸如校园网或公司网这类的本地网络中使用代理服务器的例子。代理服务器安装在本地网络中。当任意客户（浏览器）创建一个 HTTP 请求，请求首先到达代理服务器。如果代理服务器已经有响应的网页，那么它发送响应给客户。否则代理服务器会作为客户并且发送这个请求给因特网中的网络服务器。当响应返回，代理服务器在将其发送到客户端前，制作一个副本并存储到它的高速缓存中。

缓存更新

一个非常重要的问题是一个响应在被删除或被替换前，应该在代理服务器保持多长时间。很多不同的策略用来达到这个目的。一个解决方法是保存站点列表，这些站点的信息保存一段时间。比如，一个新的代理可能每天早上改变它的新闻页面。这意味着，代理服务器可以在早上获得新闻并保持它直到第二天。另一个建议是加入一些头部来显示信息的最近修改时间。代理服务器可以使用这些头部的信息去猜测它们在多长时间内是有效的。

HTTP 安全

HTTP 本质上并不提供安全。然而，我们在第 10 章指出，HTTP 可以在安全套接层（SSL）上运行。在这种情况下，HTTP 称为 HTTPS。HTTPS 提供保密性、客户和服务器鉴别，以及数据完整性。

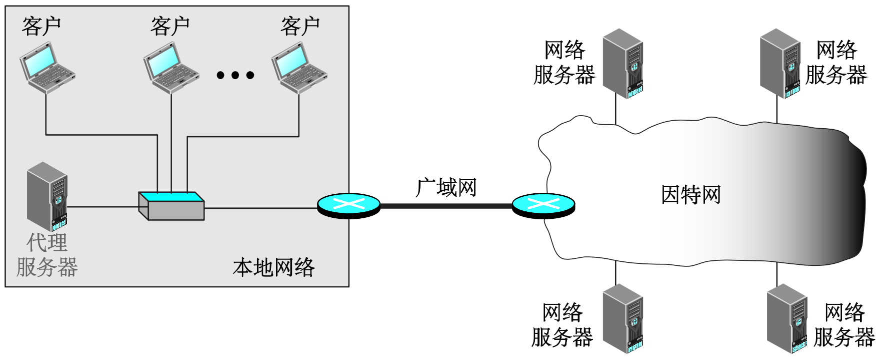


图 2-16　代理服务器的例子

2.3.2　FTP

文件传输协议（File Transfer Protocol，FTP）是 TCP/IP 提供的标准机制，用于将文件从一个主机复制到另一个主机。虽然从一个系统到另一个系统传送文件看起来很简单而且直观，但首先还是要解决一些问题。例如，两个系统可能使用不同的文件名约定。两个系统使用不同的方法来表示文本和数据。两个系统具有不同的目录结构。所有这些问题都已经由 FTP 以一种非常简单而巧妙的方法解决了。尽管我们可以使用 HTTP 传送文件，但是 FTP 是传送大文件或使用不同格式传送文件的更好选择。图 2-17 展示了 FTP 基本模型。客户有三个组件：用户接口、客户控制进程和客户数据传输进程。服务器有两个组件：服务器控制进程和服务器数据传输进程。控制连接是在控制进程之间进行的，而数据连接是在数据传输进程之间进行的。

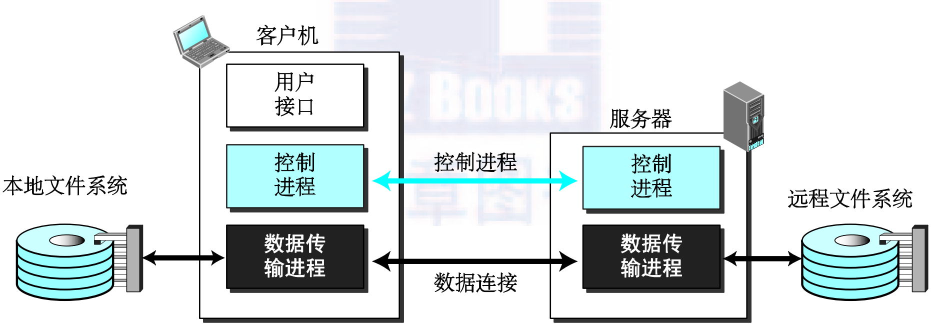


图 2-17　FTP

将命令和数据传输分开使得 FTP 效率更高。控制使用非常简单的通信规则。我们一次需要传输的只是一行命令或者一行响应。另一方面，因为传输的数据类型种类多，数据传输需要更加复杂的规则。

两种连接的寿命

FTP 中的两种连接有不同的寿命。在整个交互的 FTP 会话期间，控制连接始终处于连接状态。数据连接则在每次传输文件时开启然后关闭。每当涉及文件传输的命令被使用时，数据连接就被打开，而当文件传输完毕时连接就关闭。换言之，当用户开始 FTP 会话时，控制连接就被打开。在控制连接处于打开状态期间，如果传输多个文件，那么数据连接可以打开和关闭多次。FTP 使用两个熟知端口：端口 21 用于控制连接，端口 20 用于数据连接。

控制连接

对于控制通信，FTP 使用与 TELNET（后文讨论）相同的方法。它与 TELNET 一样使用 NVT ASCII 字符集。通信是通过命令和响应来完成的。这种简单方法适合控制连接。因为我们一次发送

一条命令（或响应）。每一条命令或响应都是一个短行，因此不必担心它的文件格式或文件结构，每一行结束处是两个字符（回车和换行）的行结束标记。

在控制连接期间，命令从客户端发送到服务器并且响应从服务器发送到客户端。从 FTP 客户控制进程发送的命令是 ASCII 大写字母形式的，可能带有也可能不带有参数。一些常见命令如表 2-4 所示。

表 2-4　一些 FTP 命令

命　令	参　数	说　明
ABOR		放弃之前的命令
CDUP		改变到上级父目录
CWD	目录名	改变到另一个目录
DELE	文件名	删除文件
LIST	目录名	列出子目录或文件
MKD	目录名	创建新目录
PASS	用户密码	密码
PASV		服务器选择一个端口
PORT	端口标识符	客户选择一个端口
PWD		显示当前目录名
QUIT		退出登录
RETR	文件名	获取文件；文件从服务器发送到客户
RMD	目录名	删除目录
RNFR	文件名	标识一个将被重命名的文件
RNTO	文件名（新）	重命名文件
STOR	文件名	存储文件；文件从客户发送到服务器
STRU	F、R 或 P	定义数据组织（F：文件，R：记录，或者 P：页面）
TYPE	A、E、I	定义文件类型（A: ASCII，E: EBCDID，I：图像）
USER	用户 ID	用户信息
MODE	S、B 或 C	定义传输方式（S：流，B：块，C：压缩）

每个 FTP 命令至少产生一个响应。一个响应有两部分：跟随在文本后的一个三位数字，数字部分定义了编码；文本部分定义了需要的参数或进一步的解释。第一个数字定义了命令状态。第二个数字定义了状态应用的区域。第三个数字提供了额外信息。表 2-5 给出了一些常见响应。

表 2-5　一些 FTP 响应

编　码	说　明	编　码	说　明
125	数据连接打开	250	请求文件动作成功
150	文件状态良好	331	用户名成功；需要密码
200	命令成功	425	无法打开数据连接
220	服务就绪	450	文件动作未被采用；文件不可用
221	服务关闭	452	动作被放弃；磁盘空间不足
225	数据连接打开	500	语法错；无法识别的命令
226	关闭数据连接	501	参数或变量语法错
230	用户登录成功	530	用户未登录

数据连接

数据连接使用服务器站点的熟知端口 20。然而，数据连接的创建和控制连接是不同的。步骤如下：

1．客户，不是服务器，使用临时端口发起一个被动打开。这必须由客户完成，因为正是客户发出命令要求传输文件的。

2．客户使用 PORT 命令发送这个端口号到服务器。

3．服务器接收到端口号，使用熟知端口 20 发出主动打开并且接收临时端口号。

通过数据连接的通信

数据连接的目的和实现与控制连接是不同的。我们通过数据连接来传输文件。客户必须定义传输文件的类型、数据结构以及传输模式。在通过数据连接发送文件之前，我们通过控制连接进行准备。异构性问题可以通过定义三个通信属性来解决：文件类型、数据结构和传输方式。

数据结构　FTP 可以使用下列数据结构中的一种在数据连接上传送文件：文件结构、记录结构和页面结构。文件结构格式（默认使用）没有结构。它是连续的字节流。在记录结构中，把文件划分成一些记录。这只能用于文本文件。在页面结构中，把文件划分成页面，每一个页面有一个页面号和页面头部，页面可以随机地或顺序地存储或访问。

文件类型　FTP 可以在数据连接上传送下列文件中的一种：ASCII 文件、EBCDIC 文件和图像文件。

传输方式　FTP 可以使用下列三种传输方式之一在数据连接上传送文件：流方式、块方式和压缩方式。流方式是默认方式，数据作为连续的字节流从 FTP 传递给 TCP。在块方式中，数据可以按块从 FTP 传递给 TCP。在这种情况下，每一个块前面有一个 3 字节的头部。第一字节称为块描述符，后面两个字节以字节为单位定义块的大小。

文件传输

文件传输是在控制连接上发送出来的命令的控制下，在数据连接上进行的。然而，我们要记住，FTP 的文件传输表示三件事情之一：读取文件（服务器到客户）、存储文件（客户到服务器）和目录列表（服务器到客户）。

例 2.11　图 2-18 给出了使用 FTP 读取文件的一个例子。图中展示了只有一个文件将要传送的情况。控制连接总是保持打开，但是数据连接重复地打开和关闭。我们假设文件以六个部分传输。在所有记录都被传输后，服务器控制进程宣布文件传输完成。由于客户控制进程没有文件要读取，它发出 QUIT 命令，这导致了服务连接被关闭。

例 2.12　下面给出了一个实际的 FTP 会话，它列出目录。灰色的行表示来自服务器控制连接的响应，黑色的行表示用户发送的命令。黑底反白的行表示数据传输。

```
$ ftp voyager.deanza.fhda.edu
Connected to voyager.deanza.fhda.edu.
220 (vsFTPd 1.2.1)
530 Please login with USER and PASS.
Name (voyager.deanza.fhda.edu:forouzan): forouzan
331 Please specify the password.
Password:*********
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
227 Entering Passive Mode (153,18,17,11,238,169)
150 Here comes the directory listing.
drwxr-xr-x    2    3027    411    4096  Sep 24  2002  business
drwxr-xr-x    2    3027    411    4096  Sep 24  2002  personal
drwxr-xr-x    2    3027    411    4096  Sep 24  2002  school
226 Directory send OK.
ftp> quit
221 Goodbye.
```

FTP 安全

设计 FTP 协议的时候安全并不是一个大问题。尽管 FTP 要求密码，但是密码还是用明文（未

加密）发送，这意味着它可能被截获并被攻击者使用。数据传输连接也用明文传输数据，这是不安全的。为了安全起见，可以把安全套接层加入到 FTP 应用层和 TCP 层之间。这种情况下，FTP 称为 SSL-FTP。当我们在本章后面讨论 SSH 时，我们也会探索一些安全文件传输应用。

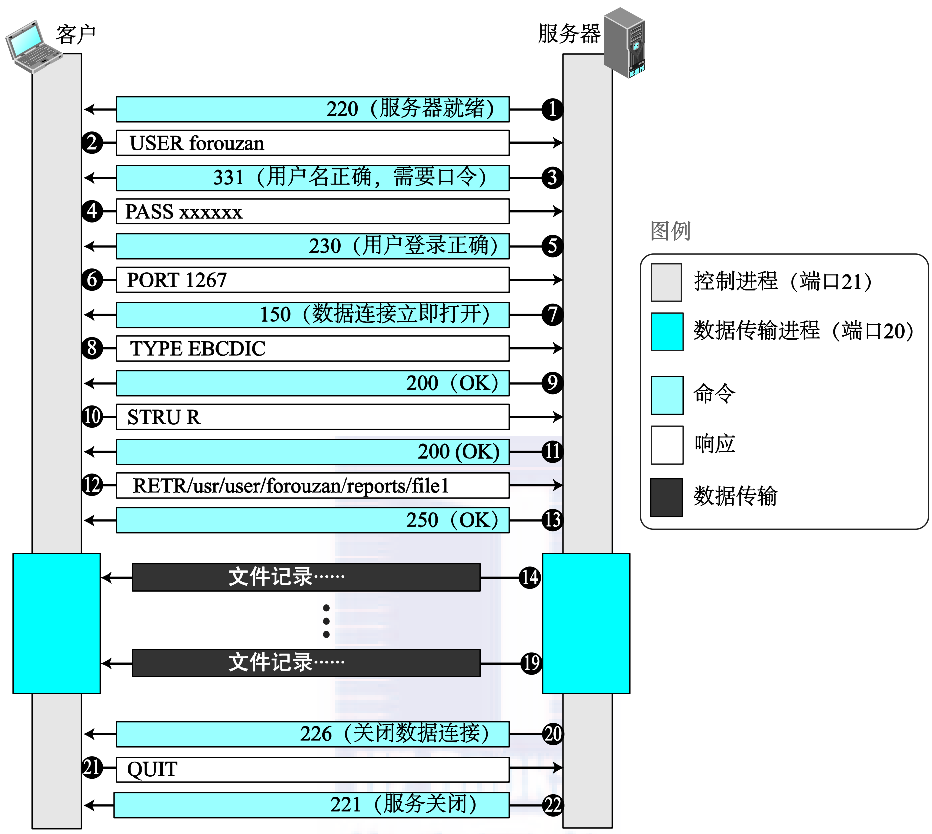


图 2-18　例 2.11

2.3.3　电子邮件

电子邮件（或 e-mail）允许用户交换报文。然而，这种应用的本质与到目前为止讨论的其他应用不同。在诸如 HTTP 或 FTP 应用中，服务器程序不断运行，等待来自客户的请求。当请求到达时，服务器提供服务。这里存在一个请求和一个应答。在电子邮件中，情况不同。首先，电子邮件是一个单向事务。当 Alice 发送一个电子邮件给 Bob，她可能期待着回复，但是这并不是一道命令。Bob 可能响应也可能不响应。如果他响应，这又是一个单向事务。第二，Bob 运行一个服务器程序并且等待别人给他发送电子邮件是不可行的也是不合情理的。当 Bob 不使用计算机时他可能将其关闭。这意味着客户/服务器编程的思想应该按照另一种方式执行：使用介于中间的计算机（服务器）。当用户想要发送邮件的时候他只运行客户程序并且中间服务器提供客户/服务器模式，这就像我们在下一节将要讨论的一样。

架构

为了说明电子邮件的架构，我们给出如图 2-19 所示的常见情景。还有一种可能是 Alice 或 Bob 直接连接到相应的邮件服务器上，那样不要求 LAN 或 WAN 连接。但是这种变化不影响我们的讨论。

在通常情景下，电子邮件的发送者和接收者，分别是 Alice 和 Bob，通过 LAN 或者 WAN 连接到两个邮件服务器上。管理员为每个用户创建了一个邮箱，接收到的报文被存储在邮箱里。邮箱是

服务器硬盘的一部分，是一个带有限制的特殊文件。只有邮箱的拥有者才能访问它。管理员也创建了一个队列（池）来存储等待发送的报文。

如图 2-19 所示，从 Alice 到 Bob 的一封简单的电子邮件需要九个不同步骤。Alice 和 Bob 使用三个不同的代理：**用户代理**（User Agent，UA）、**报文传输代理**（Mail Transfer Agent，MTA）以及**报文访问代理**（Message Access Agent，MAA）。当 Alice 需要给 Bob 发送报文时，她运行客户代理程序准备报文并发送到她的邮件服务器。然而使用 MTA 的话，报文需要从 Alice 的站点穿过因特网发送到 Bob 的站点。这里需要两个报文传输代理：一个客户和一个服务器。像因特网的客户–服务器程序，服务器需要始终运行，因为它不知道用户何时会请求一个连接。另一方面，当队列中有发送报文时，可由系统激活客户程序。Bob 处的客户代理允许 Bob 读取报文。稍后 Bob 使用一个 MAA 客户从在第二个服务器上运行的 MAA 服务器中读取报文。

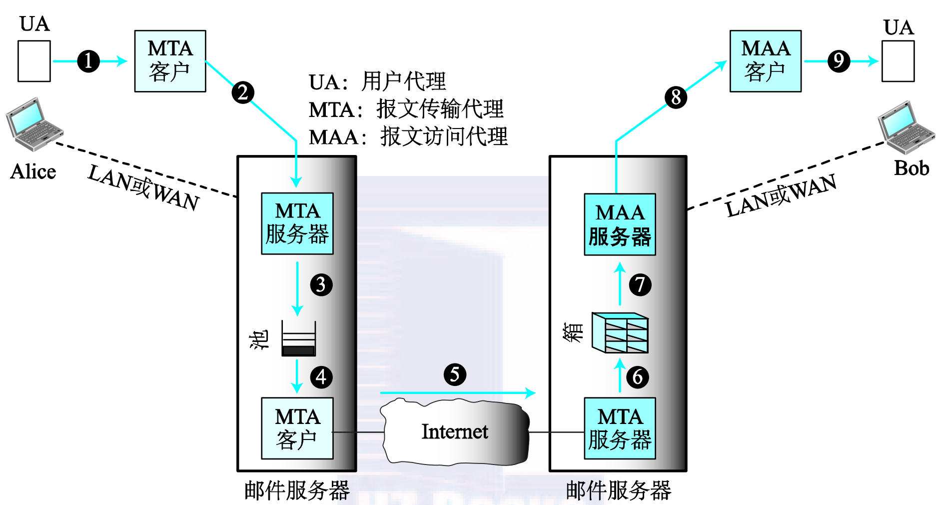


图 2-19　常见情景

我们这里需要强调两个重点。第一，Bob 不能绕开邮件服务器直接使用 MTA 服务器。为了直接使用 MTA 服务器，Bob 应该需要始终运行 MTA 服务器程序，因为他不知道报文何时到达。这意味着如果通过局域网连接他的系统，Bob 必须使计算机保持开机状态。如果他通过广域网连接到他的系统，他必须始终保持连接状态。今天，这两种情形已经都不适用了。

第二点，注意到 Bob 需要另一对客户–服务器程序：报文访问程序。这是因为 MTA 客户–服务器城区是一个推（push）程序。客户需要从服务器拉出报文。我们马上会更多地讨论 MMA。

子邮件系统需要两个 UA，一对 MTA（客户与服务器）以及一对 MAA（客户与服务器）

用户代理

电子邮件系统的第一个组件是**用户代理**（UA）。它向用户提供服务，使发送和接收一个报文变得容易。用户代理是一个软件包（程序），它由读写、回答和转发报文组成。它也处理用户计算机的本地邮箱。

有两种类型的用户代理：命令驱动型和基于 GUI 的。命令驱动型用户代理属于早期的电子邮件。在服务器中仍然存在这种类型的用户代理。命令驱动型用户代理通常是从键盘接收单个字符的命令以执行某项任务。例如，用户可以在命令提示符输入字符 r 回答报文的发送方，或输入 R 回答发送方和所有的接收者。命令驱动型用户代理的例子有：mail、pine 和 elm。

现在的用户代理都是基于 GUI 型的。它们包含图形用户接口（GUI）组件，该组件允许用户使用键盘和鼠标与软件进行交互。它们还有一些图形组件，如图标、菜单条和使服务更容易访问的

窗口。基于 GUI 的用户代理有 Eudora 和 Outlook。

发送邮件

为了发送邮件，用户通过 UA 创建邮件，邮件看起来像邮政邮件，它有一个信封和一个报文（见图 2-20）。信封通常包含发信人的地址和收件人的地址以及其他信息。报文包含头部和主体。报文头部定义了发信人、收件人、报文的主题以及其他信息。报文的主体包含了收信人要读取的真正信息。

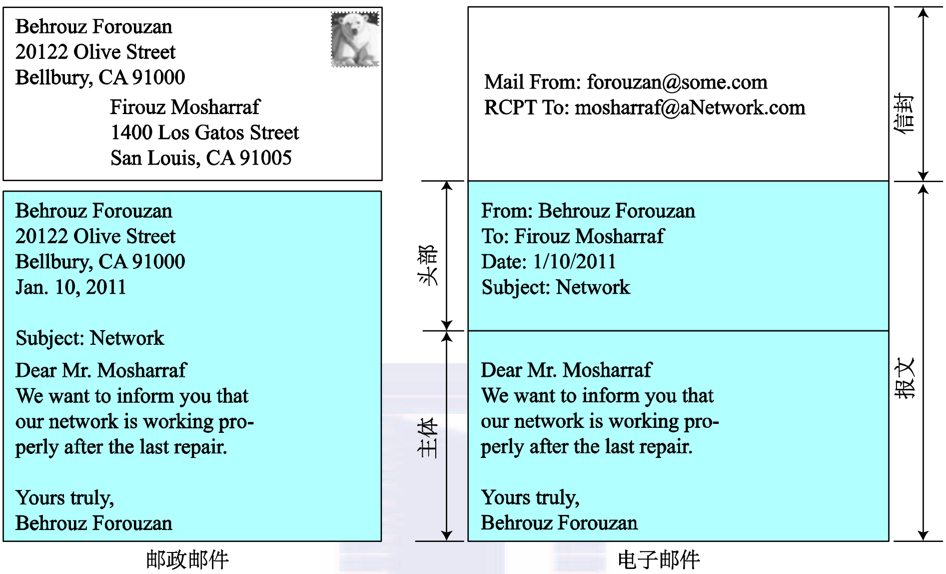


图 2-20　电子邮件格式

接收邮件

用户（或定时器）触发用户代理检查邮箱。如果用户有邮件，UA 就用一个通知来告诉用户。如果用户准备读取邮件，则会显示一个列表，其中列表的每一行包含了邮箱中关于一个特定报文的信息概要。这个概要通常包括发信人的邮件地址、主题以及发送和接收此邮件的时间。用户可以选择任何一个报文，在屏幕上显示它的内容。

地址

为了传递邮件，邮件处理系统必须使用具有唯一地址的寻址系统。在因特网中，地址由两部分构成：本地部分（local part）和域名（domain name），并且用符号@隔开（见图 2-21）。

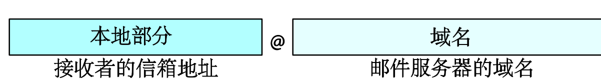


图 2-21　电子邮件地址

本地部分定义了一个特定文件的名字，它称为用户邮箱，在用户邮箱中存储了用户所有接收到的邮件，以便用户代理进行读取。第二部分是域名。一个组织机构通常选择一个或者多个主机用于接收和发送邮件；这些主机有时称为邮件服务器或交换机。分配给每一个邮件交换机的域名或者是来自 DNS 数据库，或者是一个逻辑名字（例如，该组织机构的名字）。

邮件列表或组列表

电子邮件允许用一个名字即别名来表示多个不同的电子邮件地址，这称为邮件列表。每当发送一个报文时，系统就将收信人的名字与别名数据库进行比较；如果所定义的别名有一个邮件列表，那么就将报文分开，每一个对应列表中的一项，然后交给 MTA 处理。

报文传输代理：SMTP

基于普通情景（见图 2-19），我们可以说电子邮件是那些需要使用三组客户-服务器模式完成任务

的众多应用之一。很重要的是，当我们处理电子邮件时我们要区分这三组。图 2-22 给出了这三组客户-服务器应用。我们把第一组和第二组称为报文传输代理(MTA)，第三组称为报文访问代理(MAA)。

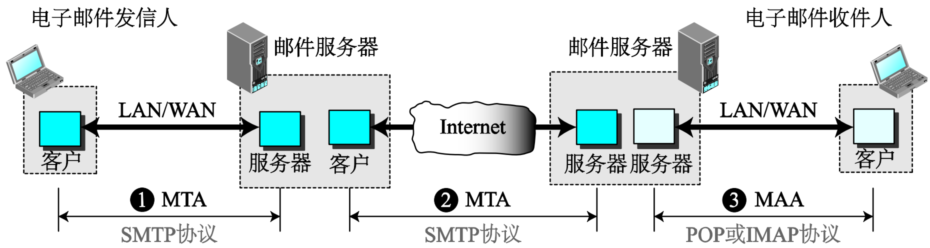


图 2-22　电子邮件中使用的协议

在因特网中定义 MTA 客户机和服务器的常用协议称为**简单邮件传输协议**（SMTP）。SMTP 在发送方和接收方邮件服务器之间以及两个邮件服务器之间被使用两次。稍后我们将看到，在邮件服务器与接收器之间还需要另一个协议。SMTP 只定义了如何来回发送命令和响应。

命令和响应

SMTP 使用一些命令和响应在 MTA 客户和 MTA 服务器之间传输报文。命令是从 MTA 客户发送到 MTA 服务器；响应是从 MTA 服务器发送到 MTA 客户。每一个命令或响应都以一个双字符(回车和换行) 的行结束标记来终止的。

命令　命令是从客户发送给服务器的，命令的格式如下：

关键词：变量（可能多个）

它包含一个关键词，后面跟着零个或多个变量。SMTP 定义了 14 个命令，如表 2-6 所示。

表 2-6　SMTP 命令

关　键　词	变　　量	说　　明
HELO	发送方的主机名	识别自身
MAIL FROM	发信人	识别发信人
RCPT TO	预期的收信人	识别收信人
DATA	邮件主体	发送实际报文
QUIT		结束报文
RSET		放弃当前邮件事务
VRFY	收信人名字	验证收信人地址
NOOP		检测收信人状态
TURN		交换发信人和收信人
EXPN	邮件列表	要求收信人扩展邮件列表
HELP	命令名	要求收信人以参数形式发送关于命令的信息
SEND FROM	预期的收信人	指定邮件只发送到收信人的终端而不是邮箱
SMOL FROM	预期的收信人	指定邮件被发送到收信人的终端或邮箱
SMAL FROM	预期的收信人	指定邮件被发送到收信人的终端和邮箱

响应　响应是服务器发给用户的。响应是一个三位数字码，后面可以跟着附加的文本信息。表 2-7 列出了各种响应。

邮件传输阶段

传输一个邮件共有三个阶段：连接建立、邮件传输和连接终止。

连接建立　在客户创建了到端口 25 的 TCP 连接后，SMTP 服务器开始连接阶段。这个阶段涉

及三个步骤：

1．服务器发送代码 220（服务就绪）告知客户它准备好接收邮件。如果服务器没有就绪，它发送代码 421（服务不可用）。

表 2-7　响应

代　　码	说　　明
肯定的完成答复	
211	系统状态或帮助回答
214	帮助报文
220	服务就绪
221	服务关闭传输通道
250	请求命令完成
251	用户不是本地的，报文将被转发
肯定中间的服务	
354	开始邮件输入
瞬态否定的完成答复	
421	服务不可用
450	邮箱不可用
451	命令异常终止；本地错误
452	命令异常终止；存储空间不足
永久性否定的完成答复	
500	语法错误，命令无法识别
501	参数或变量中有语法错误
502	命令未执行
503	命令序列不正确
504	命令暂时未执行
550	命令未执行，邮箱不可用
551	用户不是本地的
552	所请求的动作异常停止，超出存储位置
553	所请求动作未发生，不允许使用邮箱名
554	事务失败

HZ BOOKS 华章图书

2．客户发送 HELO 报文使用自身域名地址来识别自身。在这一步将客户的域名告知服务器是必要的。

3．服务器以代码 250（请求命令完成）响应，或者其他依情况而定的代码。

邮件传输　在 SMTP 客户和服务器的连接建立后，单个报文可以在一个发信人和多个收信人之间交换。这一阶段涉及八个步骤。如果有一个以上收信人，那么重复第三步和第四步。

1．客户发送 MAIL FROM 报文以介绍发信人。它包括发信人的邮件地址（邮箱和域名）。在这一步必须向发信人提供回信地址，回信地址用来返回错误以及报告报文。

2．服务器用代码 250 或其他适当的代码进行响应。

3．客户发送 RCPT TO（收信人）报文，包含了收信人的邮箱地址。

4．服务器用代码 250 或其他适当的代码进行响应。

5．客户发送 DATA 报文初始化报文传输。

6．服务器用代码 354（开始邮件输入）或其他适当的代码响应。

7．客户以连续行的形式发送报文内容。每行用两个行结束标记（回车和换行）终止。报文用

一个只包含一个句点的行终止。

8. 服务器用代码 250（OK）或其他适当的代码进行响应。

连接终止　在报文成功传送后，客户终止连接。这个阶段涉及两步。

1. 客户发送 QUIT 命令。
2. 服务器用代码 221 或其他适当的代码进行响应。

例 2.13　为了给出邮件传输的三个阶段，我们用图 2-23 给出了上文中的所有步骤。图中，在数据传输阶段，我们按照信封、头部以及主体将报文分开。注意图中的步骤在每次电子邮件传输中都重复了一遍：一次是从电子邮件发信人到本地邮件服务器，另一次是从本地邮件服务器到远程邮件服务器。本地邮件服务器在接收整个电子邮件之后可能会进行缓存并在另一个时间发送到远程邮件服务器。

报文访问代理：POP 和 IMAP

邮件传递的第一阶段和第二阶段使用 SMTP。但是，因为 SMTP 是一个推（push）协议，第三阶段不使用 SMTP。它将报文从客户推入服务器。换言之，大量数据（报文）的方向是从客户到服务器。另一方面，第三阶段需要一个拉（pull）协议，客户机必须从服务器拉出报文。大量数据的方向是从服务器到客户。因此，第三阶段使用报文访问代理协议。

目前有两种报文访问代理协议：邮局协议版本 3（POP3）和因特网邮件访问协议版本 4（IMAP4）。图 2-22 显示了这两种协议的位置。

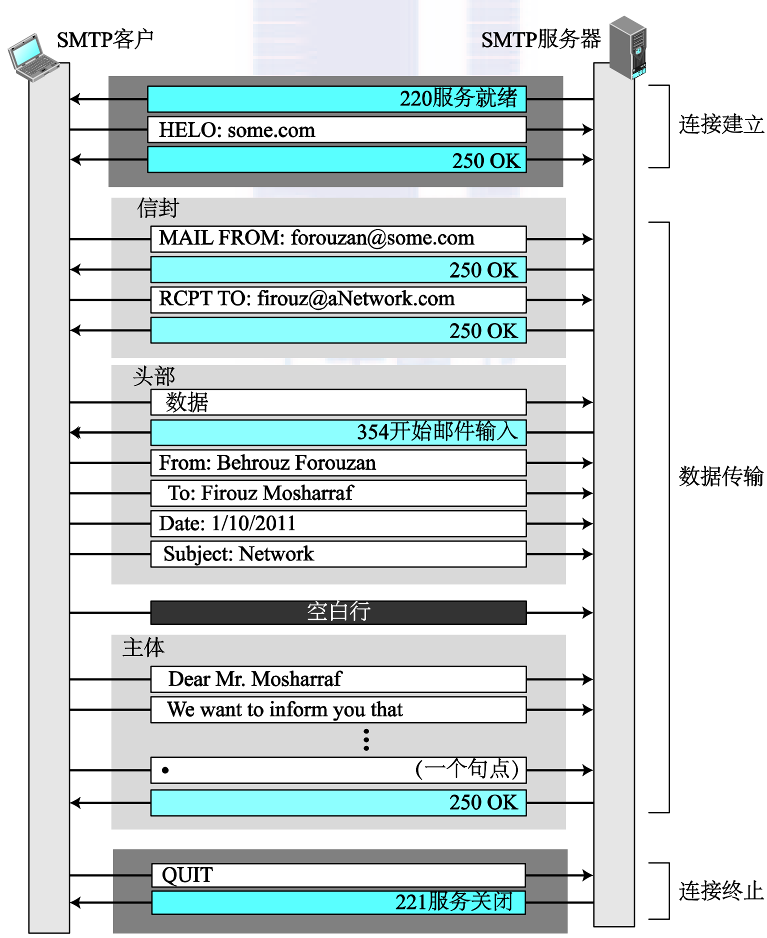


图 2-23　例 2.13

POP3

邮局协议版本 3（Post Office Protocol version 3，POP3）比较简单，但是在功能上受到一定的限制。客户端 POP3 软件安装在收信人的计算机中，服务器 POP3 软件安装在邮件服务器中。

用户需要从邮件服务器的邮箱中下载邮件时，客户端发起邮件访问操作。客户端开启与服务器 110 端口之间的 TCP 连接。然后发送用户名和口令来访问邮箱。用户就可以逐条列出和读取邮件信息了。图 2-24 说明了一个使用 POP3 下载邮件的例子。不像本章其他的图，我们将客户放在了右手边，因为电子邮件收信人（Bob）正在运行客户进程从远程邮件服务器拉报文。

POP3 有两种模式：*删除模式*和*保存模式*。在删除模式中，当邮件从邮箱中读取以后就会从邮箱中删除该邮件。在保存模式中，邮件经过读取以后仍然保存在邮箱中。当用户在固定的计算机上工作时，能够在阅读和回复邮件后保存并组织接收到的邮件，此时通常使用删除模式。当用户远离他的主计算机（如在笔记本上）访问邮件时，通常应该使用保存模式。此时可以阅读邮件，但是邮件通常仍然会保存在系统中以备日后读取和组织。

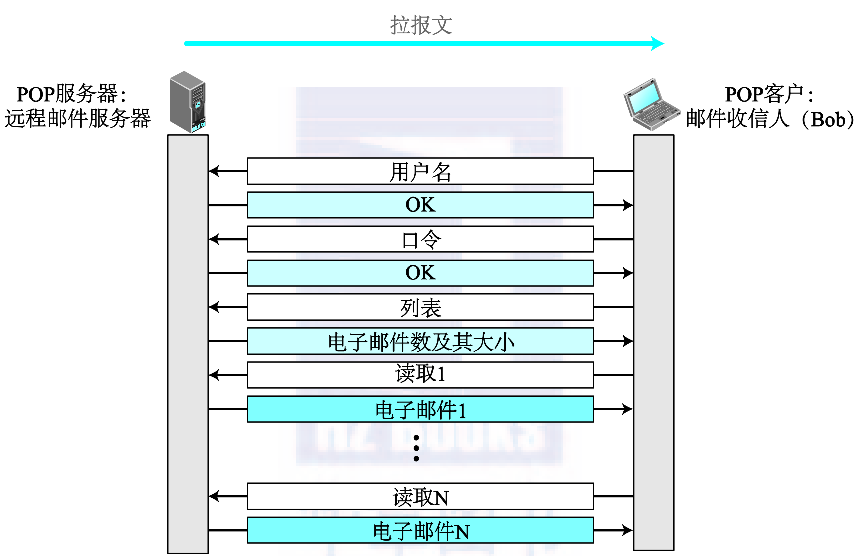


图 2-24　POP3

IMAP4

另外一个邮件访问协议是**因特网邮件访问协议版本 4**（Internet Mail Access Protocol version 4，IMAP4）。IMAP4 和 POP3 相似，但是有更多特点；功能更强并更复杂。

POP3 在很多方面存在缺点，它不允许用户在服务器上组织邮件；用户在服务器上不能有不同的文件夹。同时，POP3 不允许用户在下载邮件之前部分地查看邮件的内容。IMAP4 提供下列额外的功能：

- 用户在下载电子邮件之前，可以检查电子邮件头部。
- 用户在下载电子邮件之前，可以读取电子邮件内容中的特定字符串。
- 用户可以部分地下载电子邮件。这在带宽受限制而电子邮件包含了需要高带宽的多媒体信息时特别有用。
- 用户可以在邮件服务器上创建或删除邮箱，也可以改变邮箱的名字。
- 用户可以在文件夹中创建邮箱的层次结构以用于邮件存储。

MIME

电子邮件有一个简单的结构。但是，它的简单是有代价的。它只能发送使用 NVT 7 位 ASCII 格式的报文。换言之，它有一些限制。例如，它不能使用其他语言（如法文、德文、希伯来文、俄

文、中文以及日文)。另外，它不能用来发送二进制文件或音频以及视频数据。

多用途因特网邮件扩充（Multipurpose Internet Mail Extensions，MIME）是一个辅助协议，它允许非 ASCII 数据通过电子邮件传送。MIME 在发送方将非 ASCII 数据转换成 NVT ASCII 数据，并将其传递给 MTA 客户机通过因特网发送出去。在接收方再转换到原来的数据。

可以将 MIME 想象为一组软件功能，它能够将非 ASCII 数据转换成 ASCII 数据，以及进行相反的转换。如图 2-25 所示。



图 2-25　MIME

MIME 头部

MIME 定义了五种头部，将 MIME 头加在原来电子邮件的头部以定义转换参数：

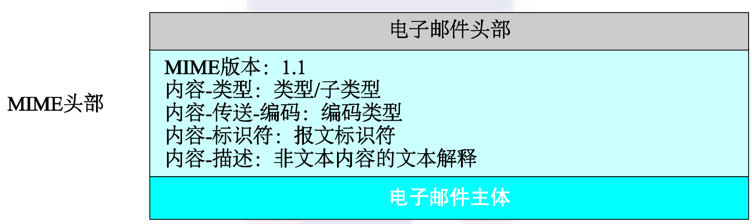


图 2-26　MIME 头部

MIME 版本　这个头部定义 MIME 使用的版本，当前版本是 1.1。

内容-类型　这个头部定义报文主体使用的数据类型。内容类型和内容子类型用一个斜杠分隔开。根据子类型的不同，头部还可以包含其他一些参数。MIME 允许七种不同的数据类型，见表 2-8。

内容-传送-编码　这个头部定义了一些方法，它们用来将传输的报文编码为 0 和 1。表 2-9 列出了五种类型的编码方法。

表 2-8　MIME 中的数据类型和子类型

类　　型	子　类　型	说　　明
正文	普通	无格式
	HTML	HTML 格式（见附录 C）
多部分	混合	主体包含不同数据类型的有序部分
	并行	同上但无序
	摘要	与混合子类型相似，但默认的是报文/RFC822
	可选择的	相同报文的不同版本部分
报文	RFC822	主体是一个封装的报文
	部分	主体是一个较大报文的一部分
	外部主体	主体是另一个报文的参照
图像	JPEG	JPEG 格式的图像
	GIF	GIF 格式的图像
视频	MPEG	MPEG 格式的视频信号

（续）

类　　型	子　类　型	说　　明
音频	基本	8kHz 的单声道话音编码
应用	PostScript	Adobe PostScript
	8 位组流	一般的二进制数据（8 位字节）

表 2-9　内容-传送-编码方法

类　　型	说　　明	类　　型	说　　明
7 位	NVT ASCII 字符，每行小于 1000 字符	Base64	6 位数据块被编码成 8 位 ASCII 字符
8 位	非 ASCII 字符，每行小于 1000 字符	引用中可打印的	非 ASCII 字符被编码成等号后面跟随一个 ASCII 码
二进制	长度不限的 ASCII 字符		

最后两个编码方法很有趣。在 Base64 编码中，数据作为位字符串，首先被分割为 6 位块，如图 2-27 所示。

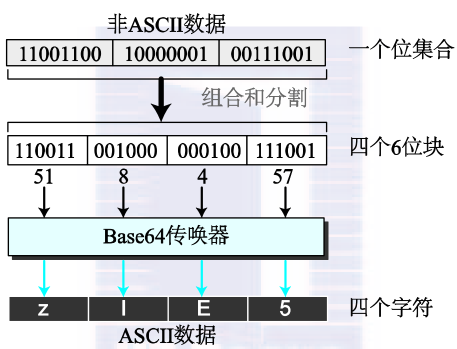


图 2-27　Base64 转换

根据表 2-10 将每个 6 位组转换成一个 ASCII 字符。

表 2-10　Base64 转换表

数值	编码	数值	编码	数值	编码	数值	编码	数值	编码	数值	编码
0	**A**	11	**L**	22	**W**	33	**h**	44	**s**	55	**3**
1	**B**	12	**M**	23	**X**	34	**i**	45	**t**	56	**4**
2	**C**	13	**N**	24	**Y**	35	**j**	46	**u**	57	**5**
3	**D**	14	**O**	25	**Z**	36	**k**	47	**v**	58	**6**
4	**E**	15	**P**	26	**a**	37	**l**	48	**w**	59	**7**
5	**F**	16	**Q**	27	**b**	38	**m**	49	**x**	60	**8**
6	**G**	17	**R**	28	**c**	39	**n**	50	**y**	61	**9**
7	**H**	18	**S**	29	**d**	40	**o**	51	**z**	62	+
8	**I**	19	**T**	30	**e**	41	**p**	52	**0**	63	/
9	**J**	20	**U**	31	**f**	42	**q**	53	**1**		
10	**K**	21	**V**	32	**g**	43	**r**	54	**2**		

Base64 是一个冗余编码方案；就是说，每 6 位成为一个 ASCII 字符并且作为 8 位来发送。我们将有 25%的开销。如果数据绝大多数是 ASCII 字符的，一少部分是非 ASCII，我们可以使用引用中可打印的编码。在引用可打印中，如果字符是 ASCII 的，它就原样发送。如果是非 ASCII，它

就作为三个字符发送。第一个字符是等号（=）。后两个字符是该字节的十六进制码。图 2-28 给出了一个例子。在例子中，第三个字符是非 ASCII，因为它是以位 1 开始的。它被解释为两个十六进制数字（$9D_{16}$），这个字符用三个 ASCII 字符代替（=、9 和 D）。

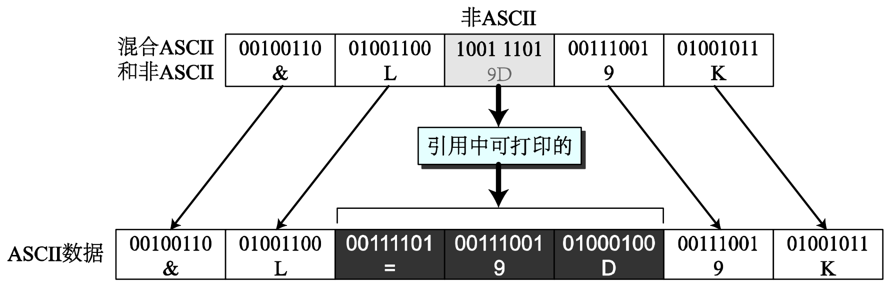


图 2-28 引用中可打印

内容-标识符 这个头部在多报文环境中唯一地标识整个报文。

内容-描述 这个头部定义主体是否为图像、音频或视频。

基于 Web 的邮件

电子邮件是一种很常见的应用，目前有一些 Web 网站对任何访问者都提供服务。两个常用的网站是 Hotmail 和 Yahoo。其思想很简单。图 2-29 给出了两种情况：

情况 1

在第一种情况中，Alice 这名发信人使用传统的邮件服务器；Bob 这名收信人在基于 Web 的服务器上有一个账户。通过 SMTP，邮件从 Alice 的浏览器传输到她的邮件服务器。报文从发送邮件服务器到接收邮件服务器的传输仍然是通过 SMTP 完成的。但是，从接收服务器（Web 服务器）到 Bob 的浏览器是通过 HTTP 完成的。换言之，HTTP 是经常使用的，而不是 POP3 或 IMAP4。当 Bob 需要取回他的电子邮件时，他发送一个 HTTP 请求报文给网站（比如 Hotmail）。网站发送一个由 Bob 填写的表格，它包括了登录名以及口令。如果登录名和口令匹配，电子邮件会从 Web 服务器以 HTML 格式传送到 Bob 的浏览器。

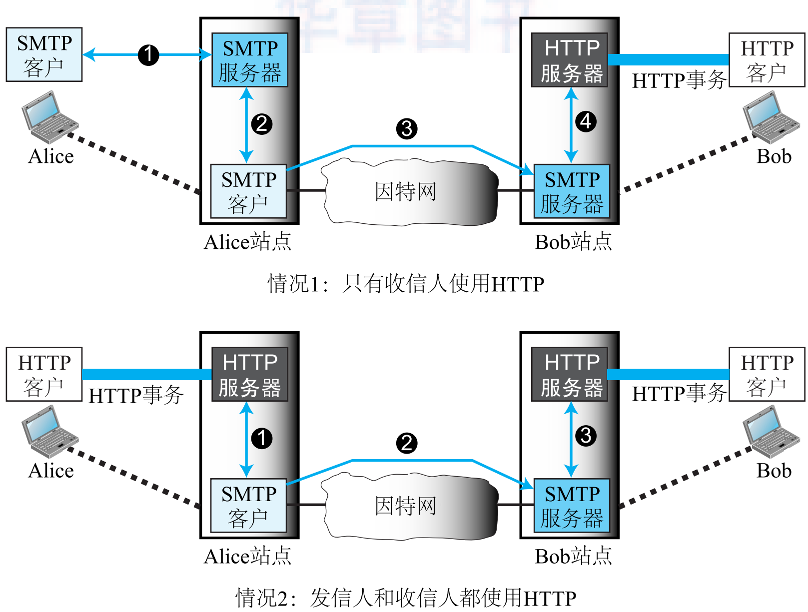


图 2-29 基于 Web 的电子邮件，情况 1 和情况 2

情况 2

在第二种情况中，Alice 和 Bob 都是用 Web 服务器，但是不必是同一个服务器。Alice 使用 HTTP 事务发送报文给 Web 服务器。Alice 使用 Bob 的邮箱作为 URL 发送一个 HTTP 请求报文给他的 Web 服务器。Alice 站点的服务器将这个报文传递给 SMTP 客户，并使用 SMTP 协议将其发送给 Bob 站点的服务器。Bob 使用 HTTP 事务接收报文。然而，从 Alice 站点服务器到 Bob 站点服务器的报文仍然使用 SMTP 协议。

电子邮件安全

本章讨论的协议其本身不提供安全。然而，可以使用两种为电子邮件系统特别设计的应用层安全来保护电子邮件的交换。我们讨论基本网络安全之后将在第 10 章讨论良好隐私（Pretty Good Privacy，PGP）以及安全多用途因特网邮件扩展（Secure/Multipurpose Internet Mail Extensions，S/MIME）。

2.3.4　TELNET

一个服务器程序可以为相应的客户程序提供特定的服务。例如，FTP 服务器被设计用来让 FTP 客户存储或获取服务器站点上的文件。然而，我们不可能对每一种所需要的服务都有一个客户-服务器对；服务的数量很快就变得难以处理。这个想法是不可扩展的。另一个解决方法是对一组常见情景使用特定的客户-服务器程序，但是要有一些通用的客户-服务器程序，这些程序允许客户端的用户登录到服务器站点上的计算机并且使用那里的可用服务。例如，如果一个学生需要使用她大学实验室的 Java 编译器服务器。这名学生可以使用一个客户登录程序登录到大学服务器并且使用大学的编译器程序。我们把这种通用客户-服务器程序对称为**远程登录**（remote login）应用。

原始的远程登录协议之一是 TELNET，这是**终端网络**（TErminaL NETwork）的缩写。尽管 TELNET 需要登录名和口令，但是它很容易遭到攻击，因为它用明文（非加密）发送所有数据包括口令。一个黑客可以偷听并获得登录名和密码。由于安全问题，TELNET 的使用让位于另一种协议——安全人机界面（Secure Shell，SHH），我们将在下一节描述这个协议。尽管 TELNET 几乎被 SSH 所替代，但是我们简要讨论 TELNET 的原因如下：

1．TELNET 的简单明文架构允许我们解释事件和一系列与登录相关的挑战，当作为远程登录协议进行服务时，这些也被用在 SSH 上。

2．网络管理员经常使用 TELNET 进行诊断和调试。

本地与远程登录

我们首先讨论本地和远程登录，如图 2-30 所示。

当一个用户登录到本地系统，这称为本地登录（local login）。用户在终端上键入时或在工作站运行终端仿真程序时，她的击键就被终端驱动程序所接受。终端驱动程序将字符传递给操作系统。操作系统解释字符的组合，并调用所需的应用程序或实用工具。

然而，当用户想访问远程机器上的一个应用程序或实用工具时，她就需要进行远程登录（remote login）。此时就需要使用 TELNET 客户程序和服务器程序。用户将击键发送给终端驱动程序，同时本地操作系统接收这些字符，但并不解释它们。这些字符被发送到 TELNET 客户机，它将这些字符转换成网络虚拟终端（Network Virtual Terminal，NVT）（下文讨论）字符的通用字符集，然后将其传送给本地 TCP/IP 协议堆栈。

采用网络虚拟终端（NVT）形式的命令或文本通过因特网传送到远程机器的 TCP/IP 堆栈，在那里字符传递给操作系统，然后传送给 TELNET 服务器，TELNET 服务器将这些字符转换成远程计算机可以理解的字符。但是，这些字符不能直接传递给操作系统，因为远程的操作系统不能接收来自 TELNET 服务器的字符，它只能接收来自终端驱动程序的字符，解决方法是增加一个称为伪

终端驱动程序（pseudo terminal driver）的软件块，它将这些字符伪装成好像是从一个终端发来的，然后操作系统将这些字符传送给适当的应用程序。

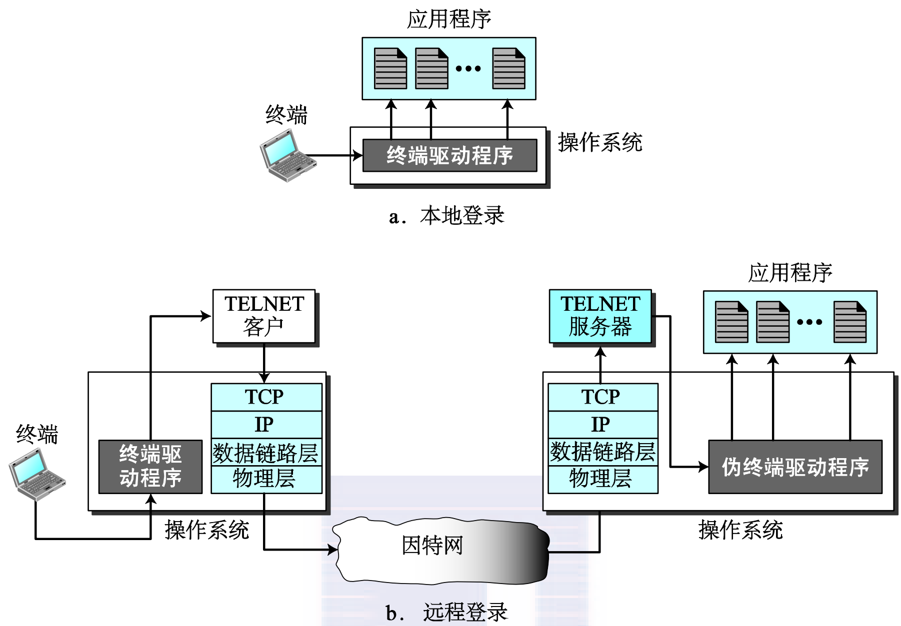


图 2-30　本地和远程登录

网络虚拟终端（NVT）

访问一个远程计算机的机制是复杂的。这是因为每一个计算机以及其操作系统都接收一种特殊的字符组合作为一个标记。例如，在运行 DOS 操作系统的计算机中，文件结束标记是 Ctrl+z，但在 UNIX 操作系统中则是 Ctrl+d。

我们现在是和异构系统打交道。如果我们想要访问世界上任何远程计算机，那么我们必须先知道将要连接的计算机是什么类型，我们还必须安装那个计算机所用的终端仿真程序。TELNET 解决这个问题的方法是定义一个通用的接口，称为**网络虚拟终端**（Network Virtual Terminal，NVT）字符集。通过这个接口，客户 TELNET 将来自本地终端的字符（数据或命令）转换成 NVT 形式，然后传递给网络。而服务器 TELNET 将来自 NVT 形式的数据或命令转换成远程计算机可接受的形式。图 2-31 表示了这一概念。

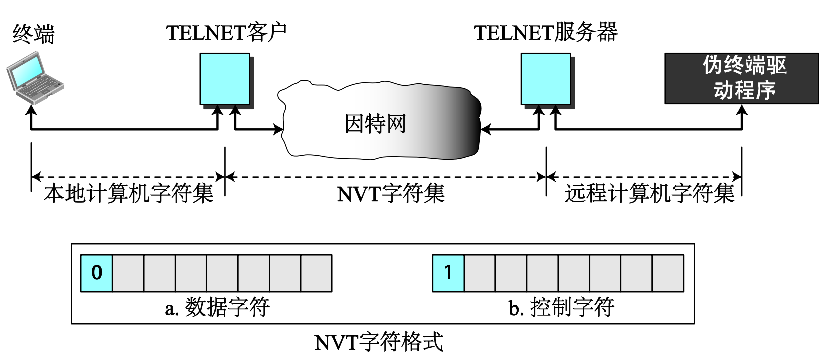


图 2-31　NVT 的概念

NVT 使用两个字符集：一个是数据字符，另一个是控制字符。如图 2-31 所示，两者都是 8 位

的字符集。对于数据，NVT 通常使用称为 NVT ASCII 的字符集。这是一个 8 位字符集，其中 7 个最低位与 ASCII 是一样的，而最高位是 0。在计算机之间发送（从客户到服务器，反之亦然）控制字符，NVT 使用一个 8 位的字符集，其最高位是 1。

选项

TELNET 允许客户与服务器在使用服务之前或使用过程中协商选项。对更加复杂的终端用户，这些选项还提供额外的特性。使用较简单终端的用户可以使用默认的特性。

用户接口

操作系统（比如 UNIX）定义了带有用户友好命令的接口。表 2-11 给出了命令集的一个例子。

表 2-11　接口命令的例子

命　令	含　义	命　令	含　义
open	连接远程计算机	Set	设置操作参数
close	关闭连接	Status	显示状态信息
display	显示操作参数	Send	发送特殊字符
mode	切换至行方式或字符方式	quit	退出 TELNET

2.3.5　安全 Shell

尽管如今的**安全 Shell**（Secure Shell，SSH）是一个可以用于远程登录和文件传输等多用途的安全应用程序，但是它原先是被设计来替代 TELNET 的。有两个版本的 SSH：SSH-1 和 SSH-2，它们是完全不兼容的。第一个版本 SSH-1 由于安全漏洞现在已被废止。在这一节，我们只讨论 SSH-2。

组件

SSH 是一个有三个组件的应用层协议，如图 2-32 所示。

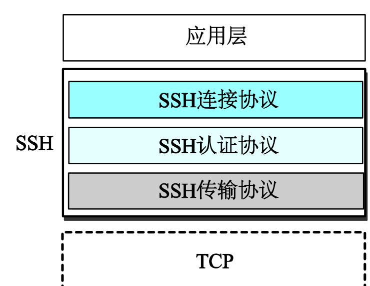


图 2-32　SSH 组件

SSH 应用层协议（SSH-TRANS）

由于 TCP 不是安全传输层协议，SSH 首先使用在 TCP 顶部创建安全链路的协议。这个新层是一个独立协议，称为 SSH-TRANS。当执行这个协议的程序被调用时，客户和服务器首先使用 TCP 协议建立一个不安全的连接。然后，它们交换几个安全参数在 TCP 顶部建立安全信道。我们将在第 10 章讨论传输层安全，但是，这里我们简要列出这个协议提供的服务：

1．隐私或交换报文的保密性。
2．数据完整性，这意味着客户和服务器交换的报文不会被入侵者改变。
3．服务认证，这意味着客户现在确认服务器正是其所声称的那台服务器。
4．报文压缩，提高了系统的效率并且使得进攻更难。

SSH 认证协议（SSH-AUTH）

在客户与服务器之间的安全信道建立以及客户端认证服务器之后，SSH 可以调用另外一个流程，它为服务器对客户进行认证。SSH 中的客户认证过程与安全套接层（SSL）的做法非常相似，我们将在第 10 章讨论这个内容。这一层定义了很多与 SSL 中相似的认证工具。认证从客户开始，客户发送一个请求报文给服务器。请求包括用户名、服务器名以及请求数据。服务器或者以一个成功报文进行响应，它确认了客户被认证；或者以一个失败报文进行响应，它表示需要以一个新的请求报文重复过程。

SSH 连接协议（SSH-CONN）

在安全信道建立以及客户和服务器相互认证之后，SSH 可以调用一个执行第三个协议的软件，那就是 SSH-CONN。SSH-CONN 协议提供的一个服务是复用。SSH-CONN 采用前两层协议创建安全信道并

允许客户在其上创建多个逻辑信道。每个信道可以用于各自不同的目的，比如远程登录、文件传输等。

应用

尽管 SSH 经常被认为是 TELNET 的替代协议，但实际上 SSH 是一个提供客户和服务器之间安全连接的通用协议。

SSH 用于远程登录

许多免费和商业应用使用 SSH 来远程登录。其中，我们可以注意到 Simon Tatham 编写的 PuTTy，它是一个可以用来远程登录的 SSH 客户程序。另外一个应用是 Tectia，可以用在多个平台上。

SSH 用于文件传输

建立在 SSH 上用于文件传输的一个应用程序是安全文件传输程序（secure file transfer program，sftp）。sftp 应用程序使用 SSH 提供的信道来传输文件。另一个常见的应用称为安全拷贝（secure copy，scp）。这个应用使用与 UNIX 拷贝命令 cp 相同的格式来拷贝文件。

端口转发

SSH 协议所提供的一个有趣服务是**端口转发**（port forward）。我们可以使用 SSH 中可用的安全信道来访问一个不提供安全服务的应用程序。如 TELNET 和简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）这类的应用，可以使用 SSH 端口转发机制的服务，这些应用将在稍后讨论。SSH 端口转发机制创建了一个隧道，属于其他协议的报文可以穿过这个隧道。由于这个原因，这个机制有时称为 SSH 隧道。图 2-33 给出了用于安全 FTP 应用的端口转发概念。

FTP 客户可以使用本地站点的 SSH 客户来与远程站点的 SSH 服务器创建安全连接。任何从 FTP 客户到 FTP 服务器的请求都被携带通过 SSH 客户和服务器提供的隧道。任何从 FTP 服务器到 FTP 客户的请求都被携带通过 SSH 客户和服务器提供的隧道。

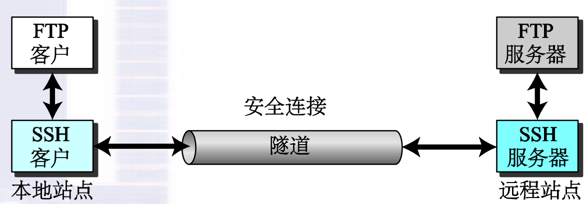


图 2-33　端口转发

SSH 分组的格式

图 2-34 给出了 SSH 协议使用的分组格式。

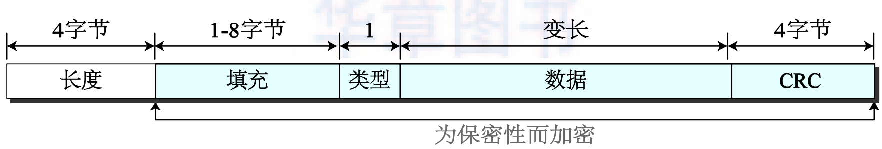


图 2-34　SSH 分组格式

长度字段定义了分组的长度但是并不包括填充字符。1 到 8 字节的填充字符被加入到分组中，使得攻击更加困难。循环冗余校验字段用于错误检测。类型字段指明在不同 SSH 协议中使用的分组类型。数据字段是不同协议中由分组传输的数据。

2.3.6　域名系统

我们最后讨论的客户-服务器程序已经被设计来帮助其他应用程序。为了识别一个实体，TCP/IP 协议使用 IP 地址唯一地确定一台主机到因特网的连接。然而，人们更喜欢使用名字而不是数字地址。所以，因特网需要一种能够将名字映射地址的目录系统。这就像电话网络。一个电话网络被设计成使用电话号码，而不是名字。人们可以保留一个私人文件将名字映射到相应的电话上或者可以让电话号码簿来做这件事情。我们将讨论这个因特网中的号码簿系统是如何将名字映射到 IP 地址的。

由于今天的因特网是如此巨大，一个中心号码簿系统不能承担所有的映射工作。此外，如果一个中心计算机失效，整个通信网络都将瘫痪。一个更好的解决方法是将信息分布在世界上的很多台计算

机中。采用这种方法，需要映射的计算机可寻找到最近一台持有所需信息的计算机。**域名系统**（Domain Name System，DNS）就是采取这种方法。本章先讨论 DNS 的概念和思想，然后描述 DNS 协议本身。

图 2-35 给出了 TCP/IP 是如何使用 DNS 客户端和服务器来将名字映射到地址的。一个用户想要使用文件传输客户端来访问运行在远程主机上相应的文件传输服务器。用户只知道文件传输服务器的名字，如 afilesource.com。然而，TCP/IP 协议簇需要文件传输服务器的 IP 地址来建立连接。以下六步将主机名映射到 IP 地址：

1．用户将主机名传递给文件传输客户端。

2．文件传输客户端将主机名传递到 DNS 客户端。

3．在启动后，每一台电脑都知道 DNS 服务器的地址。DNS 客户端使用已知 DNS 服务器 IP 地址向 DNS 服务器发送附有查询的报文，查询报文使用已知的 DNS 服务器 IP 地址给出文件服务器名。

4．DNS 服务器进行响应，发回客户想要获得的文件传输服务器的 IP 地址。

5．DNS 客户端将 IP 地址传递到文件传输服务器。

6．现在，文件传输客户端使用接收到的 IP 地址来访问文件传输服务器。

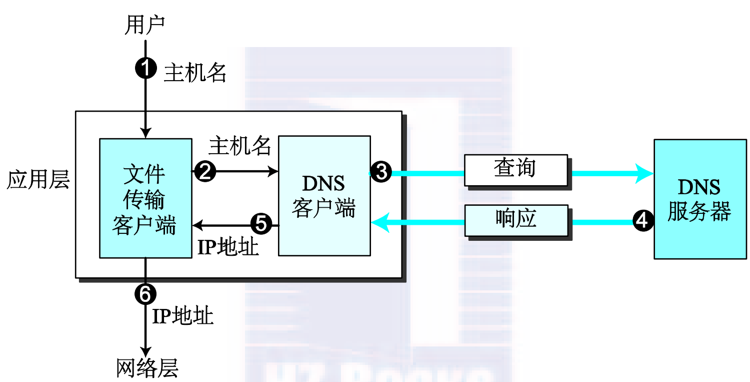


图 2-35　DNS 的用途

请注意，访问因特网的目的就是在文件传输客户端和服务器端建立连接，但是在这之前，需要在 DNS 客户端和 DNS 服务器端之间建立另一个连接。换言之，我们在这种情况下至少需要两个连接。第一个连接用于将名称映射到 IP 地址；第二个连接用于传输文件。稍后我们将看到映射可能需要一个以上连接。

名字空间

为实现无二义性，分配给机器的名字必须从名字空间中仔细选择。该名字空间完全控制对名字和 IP 地址的绑定。换言之，因为地址是唯一的，所以名字也必须是唯一的。**名字空间**（name space）将每一个地址映射到一个唯一的名字，它可以按两种方式进行组织：平面的和层次的。在*平面名字空间*（flat name space）中，一个名字分配给一个地址。空间的名字是一个无结构的字符序列。名字之间可能有也可能没有公共部分；即使有公共部分，也没有实际含义。平面名字空间的主要缺点是它必须集中控制才能避免二义性和重复，因而不能用于如因特网这样的大规模系统中。在*层次名字空间*（hierarchical name space）中，每一个名字由几个部分组成。第一部分可以定义组织的性质，第二部分可以定义一个组织的名字，第三部分可以定义组织的部门，等等。在这种情况下，分配和控制名字空间的机构就可以分散化。中央管理机构可以分配名字的一部分，这部分定义组织的性质和组织的名字。名字其他部分的分配可以交给这个组织自身。这个组织可以给名字加上后缀（或前缀）来定义主机或者其他资源。这个组织机构的管理机构不必担心为一个主机选择的后缀会被其他组织机构所采用，因为即使地址的某一部分相同，整个地址也是不同的。例如，假定两所机构把

他们的计算机都叫做 caesar。第一个机构由中央管理结构分配的名字是 first.com。第二个机构由中央管理结构分配的名字是 second.com。当每个机构在已有的名字上加上名字 caesar 后，得到了两个不同的名字：caesar.first.com 和 caesar.second.com。这些名字是唯一的。

域名空间

为了获得层次名字空间，设计了**域名空间**（domain name space）。在这种设计方式中，所有的名字由根在顶部的倒置树结构定义。该树最多有 128 级：0 级（根结点）至 127 级（见图 2-36）。

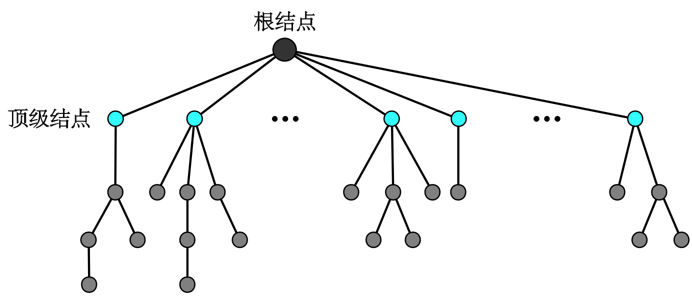


图 2-36　域名空间

标签　树上的每一个结点有一个**标签**。标签是一个最多为 63 个字符的字符串。根结点的符号是空字符串（空串）。DNS 要求每一个结点的子结点（从同一个结点分支出来的结点）有不同的标签，这样就确保了域名的唯一性。

域名　树上的每一个结点都有一个域名。一个完整的**域名**（domain name）是用点（.）分隔的标签序列。域名总是从结点向上读到根结点。最后一个标签是根结点的标签（空）。这表示一个完整的域名总是以一个空符号结束，也意味着最后一个字符是一个点（.），因为空字符串表示什么也没有。图 2-37 给出了某些域名。

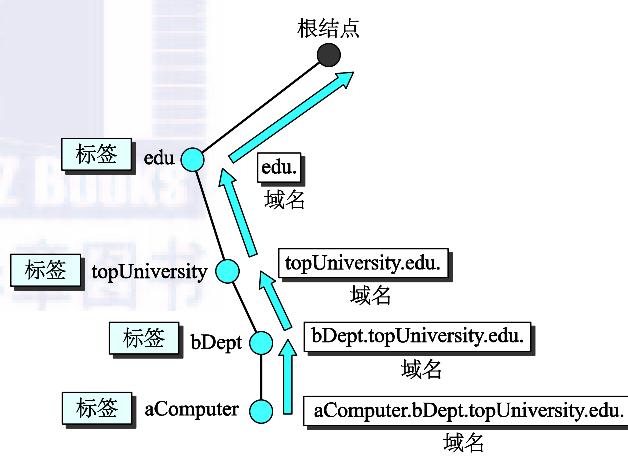


图 2-37　域名和标签

如果一个标签以一个空字符串结束，则它就称为**全称域名**（fully qualified domain name，FQDN）。名字必须以空标签结束，但是由于空标签表示没有东西，所以这种标签以一个点结束。如果一个标签不是以空字符串结束，则称为**部分域名**（partially qualified domain name，PQDN）。部分域名起始于一个结点，但没有到达根结点。当这个需要解析的名字属于和客户机相同的站点时使用部分域名。在这种情况下，解析程序能够提供省略的部分，称为后缀（suffix），以创建 FQDN。

域

域（Domain）是域命名空间的子树。域的名称是子树顶端结点的名称。图 2-38

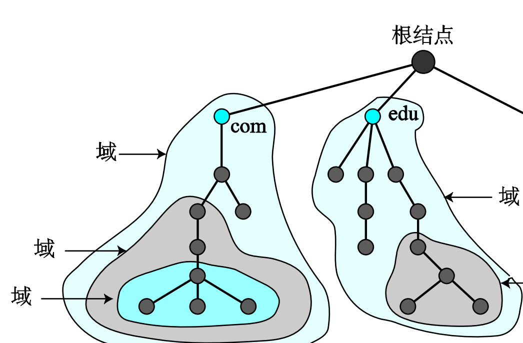


图 2-38　域

给出了一些域。注意，域本身仍可能被分割成多个域。

名字空间的分布

必须将域名空间所包含的信息存储起来。然而，只使用一台计算机存储如此大容量的信息，效率非常低和不可靠。效率低的主要原因是因为响应来自世界各地的请求会给系统造成非常大的负荷，不可靠的原因是因为任何故障将使数据无法访问。

名字服务器的层次结构　解决这些问题的办法是将信息分布在多台称为 DNS 服务器（DNS server）的计算机中。一种方法是将整个空间划分为多个基于第一级的域。换言之，让根结点保持不动，但创建许多与第一级结点一样多的域（子树）。这样创建的域会很大，DNS 允许将域进一步划分成更小的域（子域）。每一台服务器负责（授权的）一个大的域或者较小的域。换言之，与建立名字的层次结构一样，也建立了服务器的层次结构（见图 2-39）。

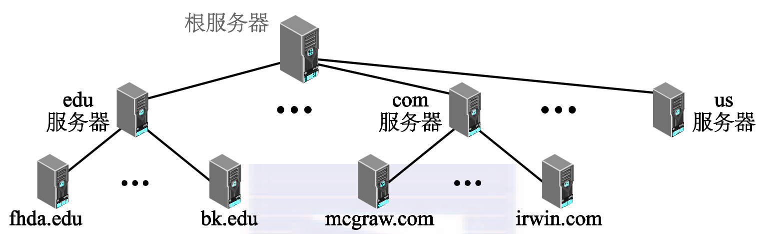


图 2-39　名字服务器的层次结构

区域

因为完整的域名层次结构不能保存在单一的服务器上，所以它被分散在多个服务器上。一个服务器负责或授权的方位称为区域（Zone）。我们可以将一个区域定义为整个树中的一个连续部分。如果服务器负责一个域，而且这个域并没有进一步被划分为更小的子域，此时“域”和“区域”是相同的。服务器有一个数据库，称为区域文件，它保存这个域里所有结点的信息。然而，如果服务器将它的域划分为多个子域，并将其部分授权委托给其他服务器，那么“域”与“区域”就不同了。在子域结点的信息会保存在较低层次的服务器中，原来的服务器则保存到这些较低层次服务器的某种参照。当然，原来的服务器并不是完全不负责任，它仍然拥有一个区域，只是将详细的信息保存在较低层次的服务器上（见图 2-40）。

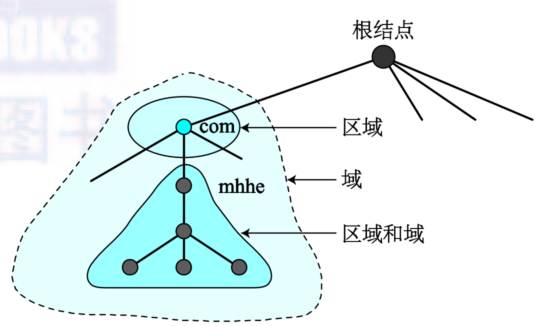


图 2-40　区域

根服务器

根服务器（root server）的区域由整棵树组成。根服务器通常不保存关于域的任何信息，只是将其授权委托给其他服务器，并保持与这些服务器的参照关系。目前有多个根服务器，每一台都覆盖了整个域名空间。这些服务器分布在世界各地。

主服务器和辅助服务器　DNS 定义了两种类型的服务器：主服务器和辅助服务器。主服务器（primary server）是指存储了授权区域有关文件的服务器。它负责创建、维护和更新区域文件，并将区域文件存储在本地磁盘中。

辅助服务器（secondary server）负责从另一个服务器（主服务器或辅助服务器）传输一个区域的全部信息，并将文件存储在它的本地磁盘中。辅助服务器既不创建也不更新区域文件。如果需要更新，则必须由主服务器来完成，由主服务器发送更新的版本到辅助服务器中。

主服务器和辅助服务器对它们所服务的区域都有控制权。这种设计思想并不是将辅助服务器置于一

个较低的授权层次上，而是为了建立数据的冗余备份，这样当一台服务器出现故障时，另一台服务器可以继续为客户机服务。注意，一台服务器可能是某个特定区域的主服务器，同时也是另一个区域的辅助服务器。所以，当提到一个服务器作为主服务器或者辅助服务器时，需要弄清楚所指的是哪个区域。

主服务器能够从磁盘文件中装载所有信息，辅助服务器从主服务器中装载信息。

因特网中的 DNS

DNS 是一种可以在不同平台上使用的协议。在因特网中，域名空间（树）被划分为三个部分：通用域、国家域和反向域。然而，由于因特网的快速增长，掌握反向域变得极其复杂，它可以用来在给定 IP 地址的情况下找到主机名。反向域现在已经废止（见 RFC 3425）。因此我们集中精力于前两个。

通用域

通用域（generic domain）按照已经注册主机的一般行为对主机进行定义。树中的每一个结点定义一个域，它是到域名空间数据库的一个索引（见图 2-41）。

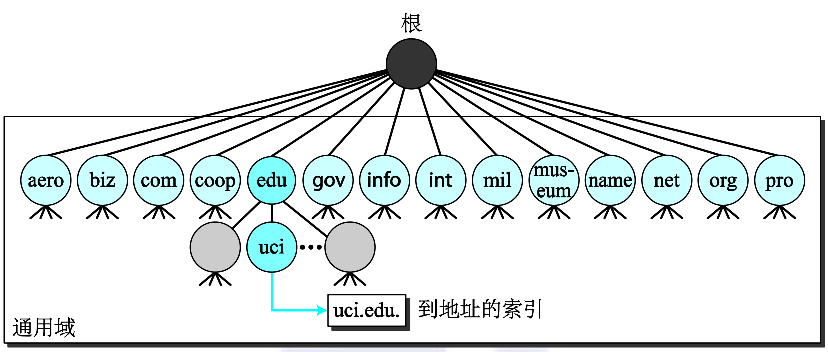


图 2-41　通用域

我们从这棵树可以看出，在通用域的一个层次允许有 14 个可能的标签。这些标签描述了表 2-12 中列出的机构类型。

表 2-12　通用域标签

标　　签	描　　述	标　　签	描　　述
aero	航空航天公司	int	国际机构
biz	商业或公司	mil	军事机构
com	商业机构	museum	博物馆
coop	协作商业组织	name	私人姓名（个人的）
edu	教育机构	net	网络支持中心
gov	政府机构	org	非盈利组织
info	信息服务提供商	pro	专业组织

国家域

国家域（country domain）部分使用双字母的国家缩写（例如，us 代表美国），第二级标号可以是机构，或者更具体一些，由各个国家自己制定。例如，美国使用州的缩写作为国家域的子域划分（例如 ca.us）。图 2-42 列出了国家域部分。地址 uci.ca.us 可以理解为美国加州的加州大学欧文分校。

解析

将名字映射为地址或者将地址映射为名字的过程，称为名字-地址解析（name-address resolution）。DNS 是一个客户/服务器应用程序。需要将地址映射为名字或者将名字映射为地址时，主机要调用一个称为**解析程序**（resolver）的 DNS 客户程序。解析程序用一个映射请求访问最近的

一个 DNS 服务器。如果服务器含有该信息，它就满足解析程序的请求；否则，它将解析程序交付给其他的服务器，或者查询其他的服务器来提供这种服务。在解析程序接收到映射后，它解释这一响应，以确定它是一个真正的解析还是一个差错，最后将结果传递给发送这一请求的进程。一个解析可能是递归的或迭代的。

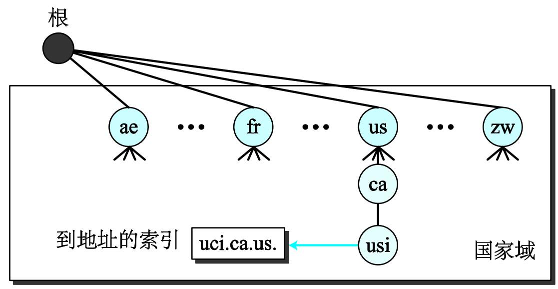


图 2-42　国家域

递归解析

图 2-43 给出了递归解析的一个简单例子。我们假设一个在名为 **some.anet.com** 的主机上运行的程序需要找到另一台名为 **engineering.mcgraw-hill.com** 的主机的 IP 地址从而发送报文。源主机被连接到 Anet ISP；目的主机被连接到 McGraw-Hill 网络。

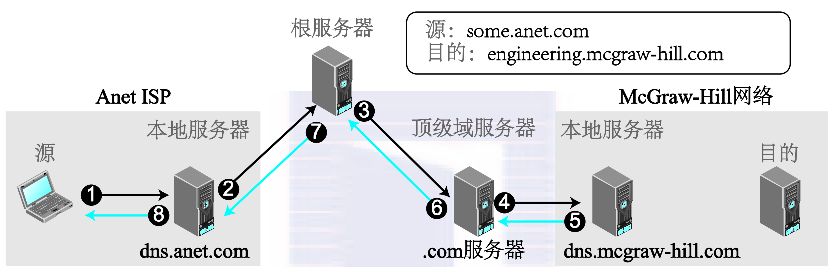


图 2-43　递归解析

源主机上的应用程序调用 DNS 解析程序（客户端）来找到目的主机的 IP 地址。解析程序不知道这个地址，便把请求发送到运行在 Anet ISP 端（事件 1）的本地 DNS 服务器（例如，dns.anet.com）。我们假设这个服务器也不知道目的主机的 IP 地址。它发送一个请求到 DNS 根服务器，本地 DNS 服务器应该知道根服务器的 IP 地址（事件 2）。根服务器通常不保存名字到 IP 地址的映射，但是根服务器至少知道每个顶级域中的一台主机（在此情况下，服务器负责 com 域）。查询被发送到这台顶级域服务器（事件 3）。我们假设这台服务器不知道这个特定目的的名字–地址映射，但是它知道本地 McGraw-Hill 公司的 DNS 服务器的 IP 地址（例如，dns.mcgraw-hill.com）。查询被发送到这台服务器（事件 4），它知道目的主机的 IP 地址。现在 IP 地址被返回给顶层 DNS 服务器（事件 5），然后返回给根服务器（事件 6），然后返回给 ISP DNS 服务器，它可能缓存这个地址以待将来的查询（事件 7），并且最终返回给源主机（事件 8）。

迭代解析

在迭代解析中，每个不知道映射的服务器将下一台服务器的 IP 地址发回到请求查询的主机上。图 2-44 给出了与图 2-43 相同场景下的迭代解析中的信息流。通常迭代解析发生在两台本地服务器之间；原始解析程序从本地服务器得到最终答案。注意事件 2、4 和 6 给出的报文包含了相同的查询。然而，事件 3 给出的报文包含顶层域服务器的 IP 地址，事件 5 给出的报文包含 McGraw-Hill 本地 DNS 服务器的 IP 地址，事件 7 给出的报文包含目的 IP 地址。当 Anet 本地 DNS 服务器接收到目的 IP 地址，它将其发送到解析程序（事件 8）。

高速缓存

每当服务器接收到查询一个不属于自己域的名字时，它需要搜索自己的数据库以查找一台服务器的 IP 地址。缩短这一查询时间能提高效率。DNS 服务器使用一种称为高速缓存（cache）的机制处理这一问题。当一个服务器向另一个服务器请求映射并得到回应时，在将该回应发送给客户端之

前，先将这一信息存储在高速缓存中。如果同一客户端或者另一个客户端请求同一映射时，它会检查其高速缓存并解决这一问题。然而，为了表明客户这一响应来自于高速缓存而不是来自于授权的信息源，该服务器会将这一响应标志为非授权性的。

高速缓存能够加快解析过程，但仍存在问题。如果同一台服务器长时间缓存一个映射，可能会发送给客户端一个过期的映射。为了防止这种情况，使用了两种技术。第一种，授权服务器总是将称为生存时间（TTL）的信息添加在映射上。生存时间定义了接收服务器可以将信息放入高速缓存的时间（以秒计）。超过这一时间，该映射就变为无效，而任何查询必须再次发送到授权服务器。第二种，DNS 要求每一台服务器对每一个映射保留一个 TTL 计数器。高速缓存会定期检查，并清除掉 TTL 已经过期的那些映射。

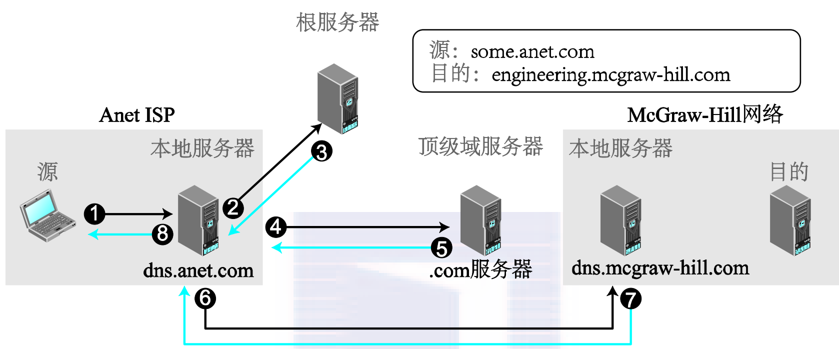


图 2-44　迭代解析

资源记录

与某个服务器相关的区域信息以资源记录（resource record）集的形式实现。换言之，域名服务器存储了资源记录的数据库。资源记录是一个 5 元组结构，如下所示：

（域名，类型，类别，TTL，数值）

域名字段标识资源记录。数值定义了保存的关于域名的信息。TTL 定义了信息有效的时间，以秒为单位。类别定义了网络的类型；我们只对类型 IN（因特网）感兴趣。类型定义了数值应当被如何解释。表 2-13 列出了常见类型以及每种类型的数值解释。

表 2-13　类型

类　型	数值的解释	类　型	数值的解释
A	一个 32 位 IPv4 地址（见第 4 章）	SOA	标记区域的开始
NS	标志区域的授权服务器	MX	转寄邮件到邮件服务器
CNAME	为主机的官方名称定义一个别名	AAAA	一个 IPv6 地址（见第 4 章）

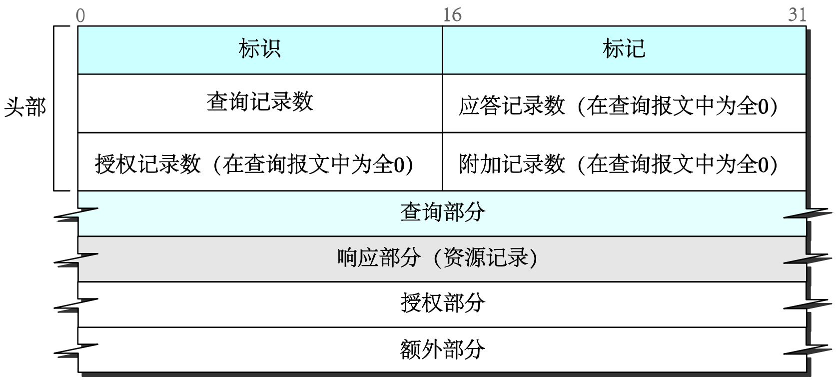
DNS 报文

为了获取关于主机的信息，DNS 使用两种类型的报文：查询报文（query）和响应报文（response）。两种类型报文都有相同的格式，如图 2-45 所示。

我们简要地讨论 DNS 报文的字段。标识字段用来匹配对查询的响应。标记定义了报文是查询报文还是响应报文。它也包含差错状态。头部接下来四个字段定义了报文中每个记录类型的数目。查询部分包含在查询报文中并且在响应中被重复，它包含一个或多个问题记录。它在查询和响应报文中都会出现。响应部分包含了一个或多于一个资源记录。它只出现在响应报文中。授权部分给出一个或多个负责查询的授权服务器的信息（域名）。额外信息部分提供了可能帮助解析程序的额外信息。

例 2.14　在 UNIX 和 Windows 系统中，nslookup 命令可以用来获取地址/域名映射。以下给出了当域名被给出时我们如何获得地址。

```
$nslookup www.forouzan.biz
Name：      www.forouzan.biz
Address：198.170.240.179
```



注：
查询报文仅仅包含查询部分。响应报文包含查询部分、响应部分，也可能包含其他两部分。

图 2-45　DNS 报文

封装

DNS 可以只用 UDP 或者 TCP 协议。在这两种情况下，服务器使用的熟知端口号是 53。当响应报文的长度小于 512 字节时，就使用 UDP。因为大多数 UDP 分组有 512 字节分组大小的限制。如果响应报文的长度大于 512 字节，则必须使用 TCP 连接。在这种情况下，可能会发生以下两种情况：

- 如果解析程序预先知道响应报文超过 512 字节，那么它必须使用 TCP 连接。例如，如果辅助名字服务器（作为客户端）需要从主服务器进行区域传输，那么必须使用 TCP 连接。因为被传输的信息通常是超过 512 字节的。
- 如果解析程序不知道响应报文的大小，那么就可以使用 UDP 端口。但是，如果响应报文超过 512 字节，那么服务器会截断这一报文。此时解析程序会开启 TCP 连接，并重复该请求，从服务器中获得完整的响应。

注册机构

新的域名是怎样加入到 DNS 中呢？这是通过注册机构（registrar）来完成的，一个熟知的商业实体是 ICANN（因特网名称和编号分配组织）。注册机构首先确认询问的域名是唯一的，然后将它输入到 DNS 数据库中，这是需要收费的。现在，有很多的注册机构，它们的名字和地址可以在如下网址中找到：

http://www.intenic.net

为了能够注册，组织机构需要给出域名服务器的主机名和 IP 地址。例如，一个名为 wonderful 的商业机构的域名服务器主机名为 ws，IP 地址为 200.200.200.5，就需要给出以下的信息给注册机构：

域名：ws.wonderful.com

IP 地址：200.200.200.5

DDNS

在设计 DNS 时，没有人预料到地址会有如此多的变化。在 DNS 中，当发生变化时，例如增加一台新主机、移动一台主机或改变一个 IP 地址时，那么这种改变就必然使 DNS 的主文件发生变化。这些类型的变化涉及手工更新。今天的因特网规模已经不允许使用这种手工操作。

DNS 主文件必须能动态更新。**动态域名系统**（Dynamic Domain Name System，DDNS）就是为满足这种需求而设计的。在 DDNS 中，当名字和地址之间的绑定被确定时，这些信息通常是由 DHCP

（见第 4 章）发送给主 DNS 服务器。主服务器更新这一区域。通知辅助服务器的方法可以以主动方式或者以被动方式。在主动通知方式中，主服务器向辅助服务器发送关于区域变化的报文；而在被动通知方式中，辅助服务器定期检查是否有任何变化。无论使用哪一种方式，当得到变化的通知时，辅助服务器会请求整个区域的信息（叫做区域传输）。

为了提供安全性以防止对 DNS 记录的非授权更改，DDNS 可以使用鉴别机制。

DNS 安全

DNS 是因特网基础设施中最重要的系统之一；它为因特网用户提供重要的服务。很多应用，例如 Web 访问或电子邮件，都强烈依赖 DNS 的正常工作。有很多方式可以攻击 DNS，这包括：

1. 攻击者可能读取 DNS 服务器的响应，从而发现用户最常访问的站点的名称。这种信息类型可以被用来找到用户信息。为了防止这种攻击，DNS 报文需要保密（见第 10 章）。

2. 攻击者可能截获 DNS 服务器的响应并加以改变，或创建一个全新的伪造响应来将用户导向攻击者想要用户访问的站点或域。这类攻击可以使用报文起源鉴别和报文完整性来进行预防（见第 10 章）。

3. 攻击者可能泛洪攻击淹没 DNS 服务器，最终使之瘫痪。这类攻击可以采取预防拒绝服务攻击的措施。

为了保护 DNS，IETF 开发了一种称为 *DNS 安全*（DNS Security，DNSSEC）的技术，它使用称为*数字签名*（digital signature）（见第 10 章）的安全服务提供报文起源鉴别以及报文完整性。然而，DNSSEC 不提供 DNS 报文机密性。在 DNSSEC 的说明中，没有针对拒绝服务攻击进行特定的保护。然而，缓存系统在某种程度上保护上层服务器免于这种攻击。

2.4　对等模式

我们在本章前面讨论了客户-服务器模式。我们在之前几节也讨论了一些标准客户-服务器应用。在这一节，我们讨论对等模式。第一个对等文件共享实例要追溯到 1987 年 12 月，当时 Wayne Bell 创建了 WWWIVnet，这是 WWIV (World War Four)公告牌软件的网络组件。在 1999 年 7 月，Ian Clarke 设计了 Freenet，一种分散、抗审查的分布数据存储，目标是通过对等网络提供带有匿名保护的言论自由。

对等随着 Napster（1999—2001）变得流行，这是一种在线音乐共享服务，由 Shawn Fanning 创建。尽管用户自由复制和分发音乐文件导致了针对 Napster 侵权法律诉讼，并最终导致关闭服务。但是它为之后到来的对等文件分发模型铺平了道路。Gnutella 在 2000 年 3 月发布了第一个发行版。紧随其后的是 FastTrack（被 Kazaa 使用）、BitTorrent、WinMX 以及 GNUnet，它们分别于 2001 年 3 月、4 月、5 月和 11 月出现。

2.4.1　P2P 网络

准备共享资源的因特网用户成为对等结点并且组成一个网络。当网络中的一个对等结点有文件（例如，一个音频或视频文件）要共享，那么它使其余对等结点都能获得这个文件。一个感兴趣的对等结点可以连接到存储这个文件的计算机并下载它。在一个对等结点下载后，它能为其他对等结点提供下载。随着越来越多的对等结点加入并下载那个文件，越来越多的文件备份可以被这个组使用。由于对等结点列表可能增长也可能萎缩，因此待解决的问题是，这个模式如何记录忠诚用户以及文件的位置。为了回答这个问题，我们首先需要将 P2P 网络分成两类：集中式与分散式。

集中式网络

在集中式 P2P 网络中，目录系统——列出对等结点及它们所提供的内容——使用客户-服务器模式，但是文件存储和下载使用对等模式完成。由于这个原因，一个集中式 P2P 网络有时称为混合 P2P 网络。集中式 P2P 网络的一个例子是 Napster。在这种网络中，一个对等结点首先在中心服

务器进行注册。之后对等结点提供自身 IP 地址以及将要分享的文件列表。为了避免系统崩溃，Napster 使用多个服务器来实现这个目的，但是我们在图 2-46 中仅给出一个服务器。

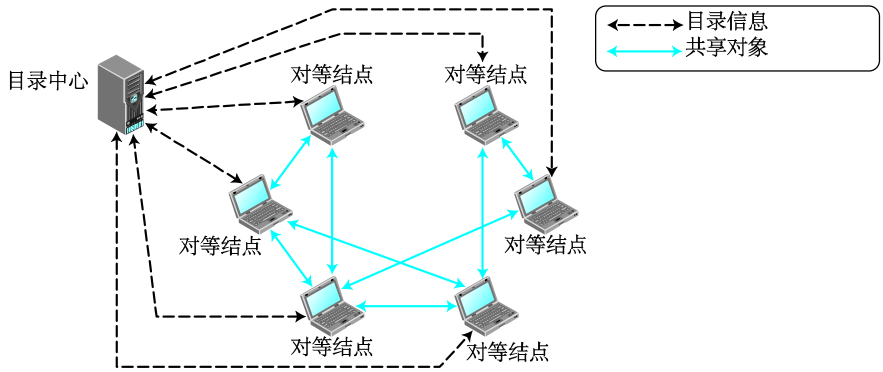


图 2-46　集中式网络

一个寻找特定文件的对等结点向中心服务器发送一个查询。服务器搜索其目录，将含有所需要文件副本的多个结点的 IP 地址作为响应返回对等结点。对等结点与其中一个结点连接并下载文件。当有结点加入或离开时，目录立即进行更新。

集中式网络对目录的维护做得很简单，但是这有很多缺点。访问目录可能产生巨大的流量并减慢系统。中心服务器易受攻击并且如果中心服务器全都失效，那么整个系统就会瘫痪。系统的中央组件是 Napster 在版权诉讼中失败并于 2001 年 7 月关闭的最终原因。Roxio 在 2003 年带来 New Napster；Napster 第二版是一个合法的、付费音乐站点。

分散式网络

分散式 P2P 网络并不依赖中心化目录系统。在这个模式中，对等结点将自身置于*覆盖网*（overlay network）中。覆盖网是一个逻辑网，在物理网的顶层创建。依照覆盖网中结点连接方式，分散式 P2P 网络分为非结构化和结构化两种。

非结构化网络

在非结构化 P2P 网络中，结点随机连接。在非结构化 P2P 中搜索效率不高，因为对一个文件的查询必须通过网络进行泛洪，这造成了极大的通信量，况且查询可能仍未解决。这类网络的两个例子是 Gnutell 和 Freenet。我们接下来把 Gnutella 作为例子来讨论。

Gnutella　Gnutella 网络是分散式非结构化对等网络的一个例子。它是非结构化的，在某种意义上目录是在结点间随机分布的。当结点 A 想要访问一个对象（例如一个文件），它联系它的一个邻居。在这种情况下，邻居是结点 A 知道地址的任意一个结点。结点 A 发送查询报文给它的邻居即结点 W。这个查询包含对象的身份（例如文件名）。如果结点 W 知道结点 X 的地址，结点 X 有这个对象，那么它就发送一个包含结点 X 地址的响应报文。结点 A 现在可以使用如 HTTP 这样的传输协议中定义的命令从结点 X 得到文件的一份副本。如果结点 W 不知道结点 X 的地址，它将请求从 A 泛洪到所有邻居。最终网络中的一个结点响应了这个询问报文，并且结点 A 可以访问结点 X。我们将在第 4 章讨论路由协议时讨论泛洪，然而此处值得注意的是，尽管 Gnutella 中以某种方式防止泛洪产生巨大的通信量负载，但是 Gnutella 不能扩展的一个原因就是泛洪。

有待解决的一个问题是，根据前文表述的过程，结点 A 是否需要知道至少一个邻居的地址。这在引导（bootstrap）时期完成，这个引导时期是结点第一次安装 Gnutella 软件的时候。软件包括一个结点（对等结点）列表，结点 A 可以将它们记作邻居。之后结点 A 可以使用两种报文，称为 ping 和 pong，来检测邻居是否仍然活跃。

正如之前所述，Gnutella 网络的一个问题是由于泛洪而缺乏扩展性。当结点数目增加时，泛洪响应不够良好。为了使得查询更高效，一个新版本的 Gnutella 使用了一种由超结点（ultra node）和叶子结点（leaf）构成的分层系统。一个进入网络的结点是叶子结点，并不负责路由；有路由能力的结点被提升为超结点。这使得查询传播得更远并且提高了效率和扩展性。Gnutella 也采取了很多其他技术，如加入了查询路由协议(Query Routing Protocol，QRP)以及动态查询(Dynamic Querying，DQ）来减少通信量开销并使搜索更高效。

结构化网络

结构化网络采用一组预先确定的规则来连接结点，有效并高效地解决查询。最常用的技术是分布式散列表(Distributed Hash Table，DHT)。DHT 应用于很多应用，包括分布式数据结构(Distributed Data Structure，DDS)、内容分发系统(Content Distributed Systems，CDS)、域名系统(Domain Name System，DNS）以及 P2P 文件共享。一种流行的使用 DHT 的 P2P 文件共享协议是 BitTorrent。我们在下一节讨论 DHT，DHT 是一种在结构化 P2P 网络和其他系统中都使用的技术。

2.4.2　分布式散列表

一个分布式散列表（Distributed Hash Table，DHT）根据预先定义的规则将数据（或引用数据）分发到一组结点上。对于基于 DHT 的网络，每一个对等结点负责一系列数据项。为了避免我们在非结构化 P2P 网络中讨论的泛洪开销，基于 DHT 的网络允许每个对等结点对整个网络做部分了解。这些对网络的了解可用于把对某数据项的查询路由到负责该资源的结点上，这个路由过程使用了我们即将讨论的一系列步骤，这些步骤是有效的并可扩展的。

地址空间

在基于 DHT 的网络中，每个数据项和对等结点都被映射到大小为 2^m 的地址上。地址空间使用模运算进行设计，这意味着我们可以把地址空间的结点想象为圆环上顺时针均匀分布的 2^m 个点（0 到 2^m–1 个），如图 2-47 所示，绝大多数的 DHT 使用 $m = 160$。

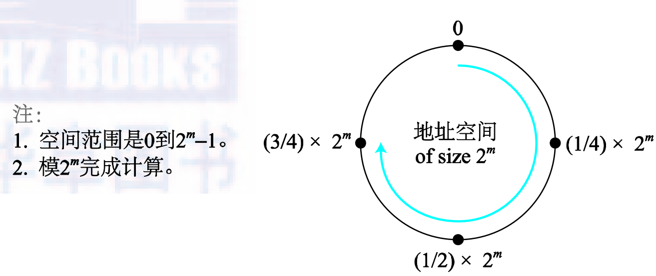


图 2-47　地址空间

散列对等结点标识符

创建 DHT 系统的第一步是将所有对等结点放入地址空间环中。这通常使用散列函数来完成，散列函数将对等结点标识符进行映射，通常是它的 IP 地址，生成一个称为结点 ID 的 *m* 位整数。

$$\text{结点 ID} = \text{hash}(\text{对等结点 IP 地址})$$

散列函数是一个从输入创建输出的数学函数。然而，DHT 使用了某些加密散列函数，例如安全散列算法（Secure Hash Algorithm，SHA），这些函数是冲突避免的。这意味着两个输入被映射到同一个输出的概率是很低的。我们将在第 10 章讨论散列算法。

散列对象标识符

被共享的对象的名称（例如一个文件）也被散列成相同地址空间上的一个 *m* 位整数。散列结果在 DHT 术语中称为关键字（key）。

$$\text{关键字} = \text{hash}(\text{对象名})$$

在 DHT 中，一个对象通常和一组（key，value）相关，其中 key 是对象名的散列值，value 是对象或对象的引用。

存储对象

存储对象有两种策略：直接方法和间接方法。在直接方法中，环中的哪个结点 ID 与对象的关键字最接近（closest），就存储在那个结点上。最接近（closest）这个术语在每个协议中定义不同。这涉及了最可能传输这个对象的电脑，这个电脑最初拥有这个对象。然而由于效率缘故，绝大多数 DHT 系统使用间接方法。拥有对象的对等结点保存对象，但是对象的引用被创建并存储在另一个结点上，这个结点的 ID 最接近关键字。换言之，物理对象以及对其引用被存储在两个不同的地点。在直接策略中，我们创建了存储对象的那个结点的 ID 与对象关键字之间的关系；在间接策略中，我们创建了对象引用（指针）与存储引用的那个结点之间的关系。在这两种情况中，如果给出对象名字就需要利用这个关系才能找到对象。在本章的其余部分中，我们使用间接方法。

例 2.15　尽管 *m* 通常为 160，但是为了演示，我们令 $m = 5$ 来使例子易于处理。在图 2-48 中，我们假设很多对等结点已经加入组。结点 N5 的 IP 地址为 110.34.56.20，它有一个名为 Liberty 的文件想要分享给它的对等结点。结点将文件名 Liberty 进行散列运算，得到 key = 14。由于最接近（closest）结点是 N17，N5 创建了对文件名（关键字）的引用、IP 地址、端口号（以及其他关于文件的信息），并发送这些引用，将其存储在结点 N17。换言之，文件存储在 N5 中，文件的关键字是 k14（DHT 环中的一个点），但是对于文件的引用存储在 N17。我们将在稍后看到其他结点如何找到 N17 并提取引用，然后使用引用来访问文件 Liberty。我们的例子仅仅给出环中的一个关键字，但在实际情况中，环中有数以百万计的关键字。

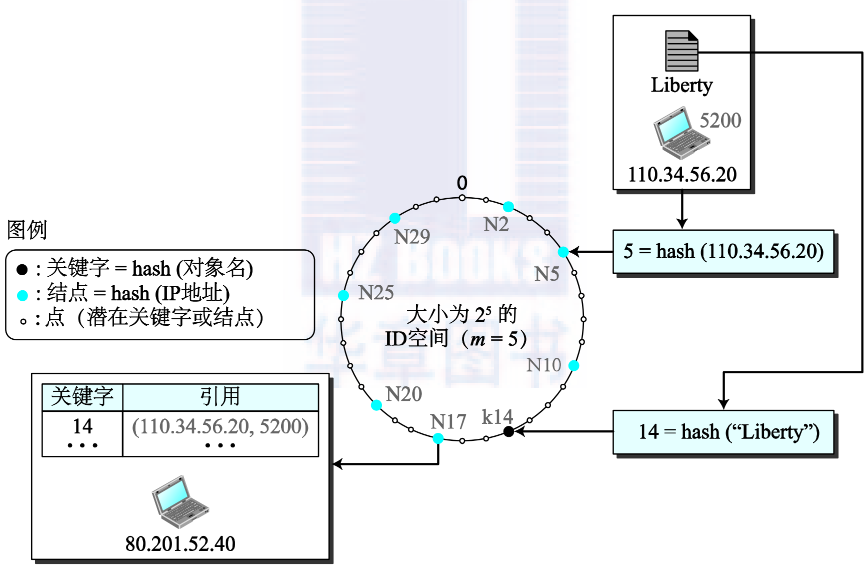


图 2-48　例 2.15

路由

DHT 的主要功能是将一个查询路由到负责存储这个对象引用的结点上。每个 DHT 的具体实现都使用不同的策略来路由，但是都依照一种思想，这种思想是每个结点必须对整个环有部分了解，从而将查询路由到与负责结点最接近的一个结点上。

结点的到达和离开

在 P2P 网络中，每个对等结点可以是一个台式机或一台笔记本电脑，对等结点可以开机或关机。当一个计算机对等结点安装了 DHT 软件，它加入了网络；当计算机关机或者对等结点关闭了软件，它就离开网络。一种 DHT 的实现需要一种清晰、有效的策略来处理结点的到达和离开，并处理对其余结点的影响。绝大多数 DHT 实现将结点失效看做结点离开。

2.4.3　Chord

有很多实现了 DHT 系统的协议。在这一节，我们介绍三种协议：Chord、Pastry 以及 Kademlia。我们选择 Chord 协议是因为它简单并且路由查询方法简约。接下来我们讨论 Pastry 协议，因为它使用与 Chord 不同的方法，并且在路由策略上与 Kademlia 协议非常接近，Kademlia 用于最流行的文件共享网络 BitTorrent。

Chord 由 Stoica 等人在 2001 年发布。我们简要介绍这种算法的主要特性。

标识符空间

Chord 中的数据项和结点是 *m* 位标识符，这些标识符创建了一个大小为 2^m 个点的标识符空间，这些点按顺时针分布在一个环上。我们将数据项的标识符称为 k（即 key，关键字），对等结点的标识符为 N（即 node，结点）。空间的数学运算是模 2^m 进行的。这意味着标识符号码在 0 到 $2^m–1$ 范围内。尽管有些实现使用了免冲突散列函数，如 SHA1 中令 $m = 160$，但是，我们在讨论中令 $m = 5$ 使得讨论更简单。最接近 N ≥ k 的对等结点称作关键字 k 的后向结点并且拥有数值（k，v），其中 k 是关键字（数据项的散列值），v 是数值（关于拥有对象的对等结点服务器的信息）。换言之，诸如文件这类数据项存储在拥有数据项的对等结点上，但是数据项的散列值 key 以及对等结点的信息 value 被作为一对（k，v）存储在 k 的后向结点上。这意味着存储数据项的对等结点和拥有（k，v）对的结点不必是同一个结点。

指针表

Chord 算法中的结点应该能够解决请求：给出一个关键字，结点应该能够找到负责这个关键字的结点标识符，或者将查询转发给另一个结点。Chord 要求每一个结点维护 *m* 的后向结点以及一个前向结点的信息。每个结点创建一个称为指针表（finger table）的路由表，如图 2-14 所示。注意到第 *i* 行的目标关键字是 $N+2^{i-1}$。

表 2-14　指针表

i	目标关键字	目标关键字的后向结点	后向结点的信息
1	N + 1	N + 1 的后向结点	后向结点的 IP 地址和端口
2	N+ 2	N + 2 的后向结点	后向结点的 IP 地址和端口
⋮	⋮	⋮	⋮
m	$N + 2^{m-1}$	$N + 2^{m-1}$ 的后向结点	后向结点的 IP 地址和端口

图 2-49 仅给出了环中的一个后向结点列，这个环仅有少量结点和关键字。请注意，实际上第一行（$i = 1$）给出了后向结点。我们也添加了前向结点的 ID，稍后我们会看到这是需要的。

接口

为了进行操作，Chord 需要一组 Chord 接口。在这一节，我们讨论部分操作来给出 Chord 协议背后的思想。

查找

Chord 中最常用的操作可能就是查找。Chord 让对等结点之间共享可用服务。为了找到被共享的对象，对等结点需要知道负责那个对象的结点：存储对象引用的对等结点。在 Chord 中，我们讨论过，环中一组关键字的后继结点就是负责那些关键字的结点。找到负责结点实际上就是找到关键字的后继结点。表 2-15 给出查找操作的代码。

查找函数使用自顶向下方法编写。如果结点负责关键字，它返回自己的 ID；否则，它调用函数 find_successor。find_successor 函数调用 find_predecessor。最终的函数调用 find_closest_predecessor。模块方法允许我们在其他操作中使用这三个函数而不必重定义它们。

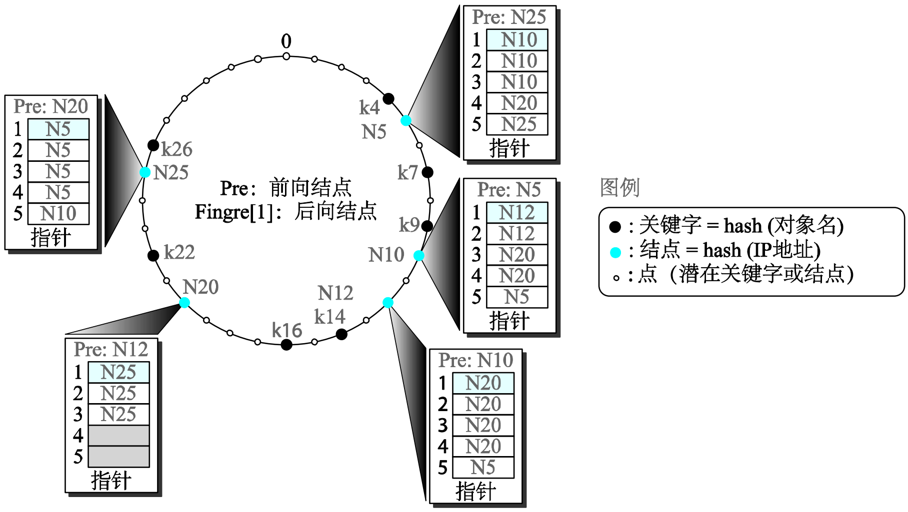


图 2-49　Chord 中环的例子

表 2-15　查找

```
Lookup (key)
{
        if (node is responsible for the key)
                return (node's ID)
        else
                return find_succesor (key)
}

find_successor (id)
{
        x = find_ predecessor (id)
        return x.finger[1]
}

find_predecessor (id)
{
        x = N                                          // N是当前结点
        while (id ∉ (x, x.finger[1]]
        {
                x = x.find_closest_predecessor (id)    // 令x寻找它
        }
        return x
}

find_closest_predecessor (id)
{
        for (i = m downto 1)
        {
                if (finger [i] ∈ (N, id))              // N是当前结点
                        return (finger [i])
        }
        return N                                       // 结点自身是最接近的前向结点
}
```

让我们来仔细研究 lookup 函数。如果结点不负责关键字 key，lookup 函数调用 find_successor 函数来找到这个 ID 的后向结点，这个 ID 是作为参数传递给 find_successor 函数的。如果我们首先找到关键字 key 的后向结点，那么后向结点函数代码可以非常简单。前向结点可以轻易地帮助我们找到环中的下一个结点，因为前向结点的第一个指针（finger[1]）给出了后向结点的 ID。此外，查找关键字前向结点的函数在其他函数中是有用的，这些函数我们稍后会编写。不幸的是，结点自己不能正常地找到前向结点；关键字 key 可能远离结点。正如我们之前讨论的，一个结点仅维护有限个其余结点的信息；指针表最多维护 *m* 个其他结点（在指针表中有一些重复数据）。由于这个原因，一个结点需要其他结点的帮助来找到关键字 key 的前向结点。这一步可以将 find_closest_predecessor 函数作为远程程序调用（remote procedure call，RPC）来完成。远程程序调用意味着调用一个在远程结点执行的函数，并且将结果返回到调用结点上。我们在算法中使用表达式 x.procedure，其中 x 是远程结点的标识符，procedure 表示被执行的程序。结点使用这个函数来找到一个更接近前向结点的点。然后，它将寻找前向结点的责任传递给其他结点。换言之，如果结点 A 想要找到结点 X，它找结点 B（最接近前向结点）并且把任务发送给 B。现在结点 B 接管控制并且试图寻找 X，或者 B 将任务发送给另一个结点 C。任务被从一个结点转发到另一个结点，直到一个结点为止，即那个结点拥有所要找的前向结点的信息。

例 2.16 假设图 2-49 的结点 N5 需要寻找负责关键字 k14 的结点。图 2-50 给出了 8 个事件的序列。在事件 4 中，find_closest_predecessor 函数返回了 N10。在事件 4 之后，find_predecessor 函数要求 N10 返回它的 finguer[1]，也就是 N12。此时，N5 发现 N10 并不是 k14 的前向结点。之后，结点 N5 要求 N10 找到最接近 k14 的前向结点，这个请求返回了 N12（事件 5 和 6）。现在，结点 N5 请求结点 N12 的 finger[1]，返回结果为 N20。现在结点 N5 进行检查，发现 N12 确实是 k14 的前向结点。这个信息被传递给 find_successor 函数（事件 7）。N5 现在请求结点 N12 的 finguer[1]，返回结果为 N20。搜索终止，N20 是 k14 的后向结点。

稳定化

在我们讨论结点如何加入和离开环之前，我们需要强调的是环中的任何改变（例如，结点、结点组的加入和到达）可能导致环不稳定。Chord 中定义的一个操作称为稳定化。环中的每个点周期性地使用这个操作来验证它的后向结点，并且令后向结点验证它的前向结点信息。结点 N 使用 finguer[1]的值 S 来要求结点 S 返回它的前向结点 P。如果从这个请求中得到的返回值 P 在 N 和 S 之间，这意味着存在一个 ID 等于 P 的结点，它位于 N 与 S 之间。结点 N 使 P 成为它的后向结点，并通知 P 使 N 成为其前向结点。表 2-16 给出了稳定化操作。

表 2-16 稳定化

```
Stabilize ()
{
        P = finger[1].Pre                          // 要求后向结点返回它的前向结点
        if(P∈ (N,finger[1]))   finger[1] = P       // P 是N的后向结点
        finger[1].notify (N)                       // 通知P改变它的前向结点
}
Notify (x)
{
            if (Pre = null or x ∈ (Pre, N))      Pre = x
}
```

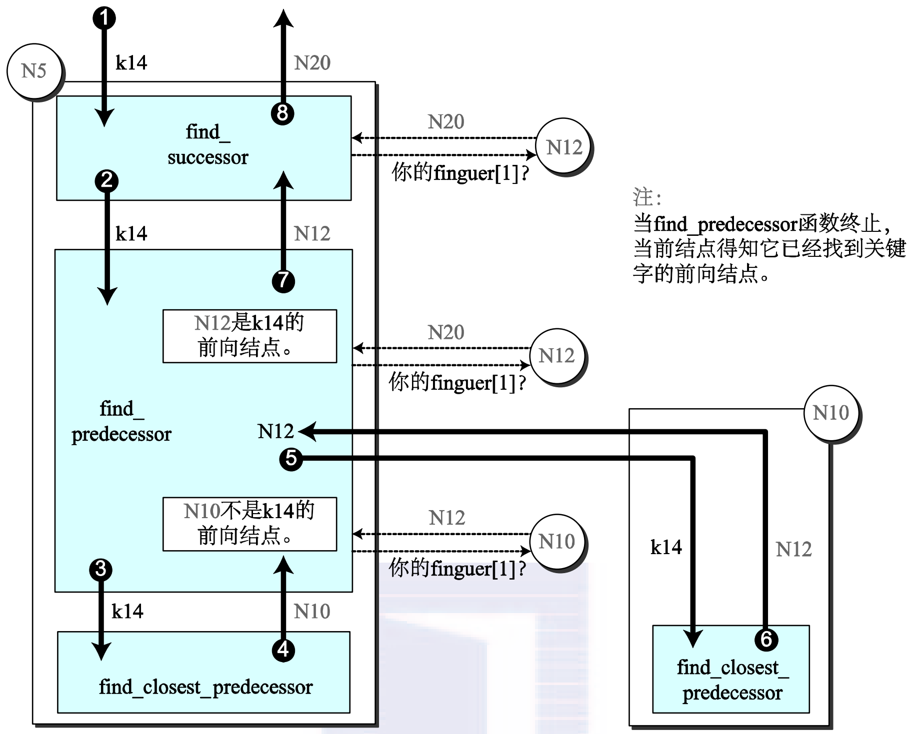


图 2-50　例 2.16

Fix_Finger

不稳定可能最多改变 m 个点的指针表。Chord 中定义的另一个操作是 fix_finger。环中的每个结点必须周期性地调用这个函数来进行指针表更新。为了减少系统的通信量，每个结点必须在每次调用中只更新它的一个指针。这个指针是随机选择的。表 2-17 给出了这个操作的代码。

表 2-17　Fix_Finger

```
Fix_Finger ()
{
        Generate (i ∈ (1, m])                   // 随机生成i， 1 < i ≤ m
        finger[i] = find_successor (N + 2^(i−1))  // 找到finger[i]的值
}
```

加入

当一个对等结点加入环，它使用 join 操作以及其他对等结点的 ID 来找到它的后向结点，并且将其后向结点置为空。它立即调用稳定化函数来验证它的后向结点。之后，结点要求后向结点调用 move-key 函数来传输新对等结点负责的关键字。表 2-18 给出这个操作的代码。

表 2-18　Join

```
Join (x)
{
        Initialize (x)
        finger[1].Move_Keys (N)
}
```

```
Initialize (x)
{
    Pre = null
    if (x = null) finger[1] = N
    else finger[1] = x. Find_Successor (N)
}

Move_Keys (x)
{
    for (each key k)
    {
        if (x ∈ [k, N)) move (k to node x)          // N是当前结点
    }
}
```

很明显，在这个操作后加入的结点的指针表是空的，并且最多有 *m* 个前向结点的指针表是过期的。在这个事件之后，周期性地运行 stablize 以及 fix_finger 操作将逐渐稳定系统。

例 2.17 我们假设图 2-49 中，结点 N17 在 N5 的帮助下加入环。图 2-51 给出了稳定后的环。过程如下：

1. N17 使用 Initialize（5）算法设置其前向结点为空，并设置后向结点（finger[1]指向 N20）。
2. 之后 N17 要求 N20 发送 k14 以及 k16 到 N17，因为 N17 现在负责这些关键字。
3. 在下一次超时，N17 使用 stabilize 操作来验证自己的后向结点（即 N20）并且要求 N20 将其前向结点改为 N17（使用 notify 函数）。
4. 当 N12 使用 stabilize 时，N17 的前向结点更新为 N12。
5. 最终，当某些结点调用 fix-finger 函数时，N17、N10、N5 以及 N12 的指针表被改变。

离开或失效

如果一个对等结点离开环或者失效（不是环失效），环将会中断运行，除非环能稳定化其自身。每个结点与邻居交换 ping 和 pong 报文来检测邻居是否还活跃。当结点没有收到回应 ping 的 pong 报文时，结点知道邻居失效了。

尽管使用 stabilize 和 fix-finger 操作可以在结点离开或失效后恢复环，但是发现问题的结点可以立即采取这些操作而不用等待超时时间。一个重要问题是当多个结点同时离开或失效时，stabilize 和 fix-finger 操作可能会不起作用。因此，Chord 要求每个结点记录 *r* 个后续结点（*r* 的值取决于具体实现）。如果一个后向结点不可用，那么总可以去往下一个后向结点。

这种情况下的另一个问题是由失效或离开的结点管理的那些数据也变得不可用了。Chord 规定只有一个结点负责一组数据和引用，但是 Chord 也规定，在这种情况下，数据和引用应该在别的结点备份。

例 2.18 在图 2-51 中，我们假设结点 N10 离开环。图 2-52 给出了稳定后的环图示。

过程如下：

1. 当结点 N5 收不到 N10 回复的 pong 报文时，N5 发现 N10 已经离开。结点 N5 改变它的后继结点（finger[1]）到 N12（后继结点列表中的第二项）。
2. 结点 N5 立即调用 stabilize 函数并要求 N12 将其前向结点改为 N5。
3. 如果顺利的话，N10 负责的 k7 和 k9 在 N10 离开前已经被复制到 N12。
4. 在若干次 fix-finger 调用之后，如图 2-52 所示，N5 和 N25 更新了它们的指针表。

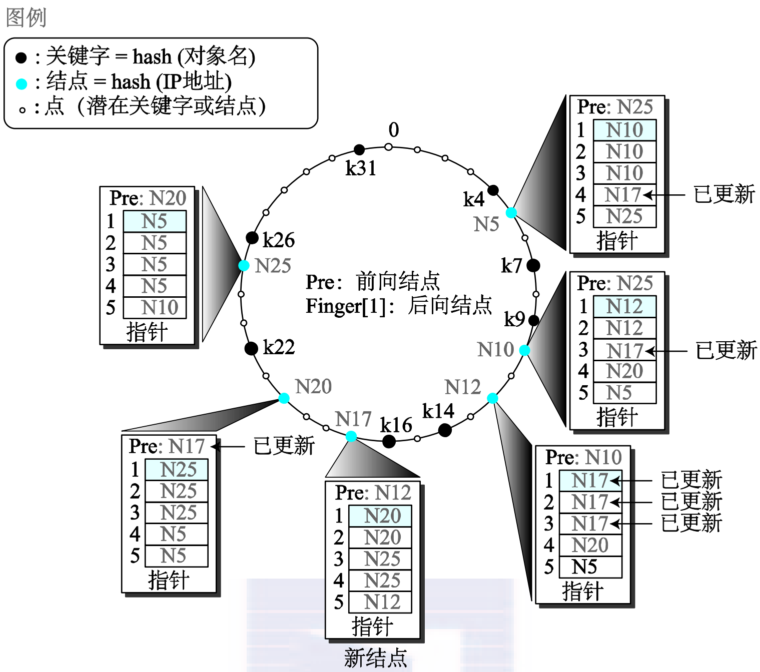


图 2-51　例 2.17

应用

Chord 有很多应用，包括协同文件系统（Collaborative File System，CFS）、ConChord 以及分布式域名系统（Distributive Domain Name System，DDNS）。

2.4.4　Pastry

另一个 P2P 模式中的流行协议是 **Pastry**，它是由 Rowstron 和 Druschel 设计的。如上所述，Pastry 使用 DHT，但是 Pastry 与 Chord 之间在标识符空间和路由过程中有一些根本性的不同，这些不同，我们将在下文介绍。

标识符空间

与 Chord 类似，在 Pastry 中的结点和数据项是一个 m 位的标识符，这些标识符创建了一个由 2^m 个顺时针均匀分布在环上的点组成的标识符空间。m 通常为 128。协议使用 SHA-1 散列算法，其中 $m = 128$。然而，在这个协议中，标识符被视为基于 2^b 的 n 位字符串，其中 b 通常为 4，并且 $n = (m/b)$。换言之，标识符是一个基于 16（十六进制）的 32 位数字。在标识符空间中，关键字存储在结点标识符与其最接近的那个结点上。这个策略与 Chord 不同。在 Chord 中，关键字存储在它的后向结点上；在 Pastry 中，关键字可能存储在数值上最接近关键字的后向或前向结点中。

路由

Pastry 结点应该能够解决查询；给定一个关键字，结点应该能够找到负责那个关键字的结点或者将查询转发到另外一个结点。Pastry 中每个结点使用两个实体来进行路由：*路由表和叶子结点集*。

路由表

Pastry 要求每个结点维护一个 n 行 2^b 列的路由表。通常，当 $m = 128$ 并且 b=4 时我们有 32（128/4）行 16 列（$2^{128} = 16^{32}$）。换言之，每一行对应标识符的一位，每一列对应十六进制的值（0 到 F）。表 2-19 给出普通情况下的路由表大纲。在结点 N 的路由表中，i 行 j 列的单元格，表[i，j]，给出了结点的标识符（如果存在的话），这个结点最左边 i 位与 N 的标识符相同，第（$i+1$）位的值为 j。

第 1 行即 0 行给出了活动结点列表，这些活动结点的标识符与 *N* 没有相同前缀。第 1 列给出另一个活动结点列表，这些活动结点与结点 *N* 最左边的 1 位数字是相同的。类似地，第 31 列给出了所有活动结点列表，这些活动结点与结点 *N* 的左边 31 位数字是相同的；只有最后一位不相同。

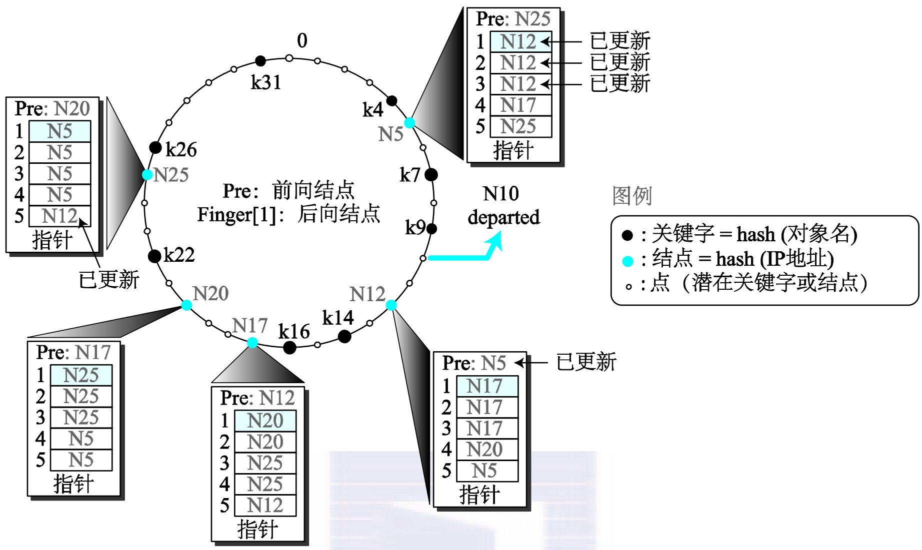


图 2-52　例 2.18

例如，如果 N = $(574A234B12E374A2001B23451EEE4BCD)_{16}$，那么表[2，D]的值可能是结点（**57D…**）的标识符。注意到最左侧两个数字为 57，与 N 的前两位数字是相同的，但是下一位数字是 D，这个数字与第 D 列相对应。如果有更多的结点带有前缀 57D，根据接近度（proximity metric），最接近的结点被选中并将其标识符插入表格内。接近度是一种由使用网络的具体应用决定的度量。它可能基于结点间的跳数、往返时间或其他度量。

表 2-19　Pastry 中结点的路由表

公共前缀长度	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
31																

叶子结点集

路由中的另一个实体是由 2^b（路由表中列的大小）个标识符组成的集合，它称为叶子结点集（leaf set）。集合中一半的结点标识符在数值上小于当前结点；集合中另一半的结点标识符在数值上大于当前结点。换言之，叶子结点集给出了环上位于当前结点之前的 2^{b-1} 个结点以及位于之后的 2^{b-1} 个结点。

例 2.19　让我们假设 m = 8 位，并且 b = 2。这意味着我们有上限为 2^m = 256 个标识符，并且每个标识符基于 2^b = 4，共有 m/b = 4 位数字。图 2-53 给出一种情形，其中一些活动结点和关键字映射到这些结点上。关键字 k1213 存储在两个结点上，因为它距离这两个结点等距。这提供了一些冗余，以防其中一个结点失效。图 2-53 也给出了四个结点的路由表和叶子结点集，这些结点在后文例子中会用到。例如，在结点 N0302 的路由表中，根据接近度，我们假设结点 1302 与结点 N0302 最近，所以选定结点 1302 插入表[0，1]。我们对于其他的表项也采取相同的策略。请注意，每个表的每一行中有一个单元格是带阴影的，这是因为它与结点标识符的数字相对应；任何其他结点标

识符都不能插入到这个单元格。有些单元格是空的，这是因为此时网络中没有满足要求的活动结点；当有新结点加入网络，它们就可以被插入这些单元格。

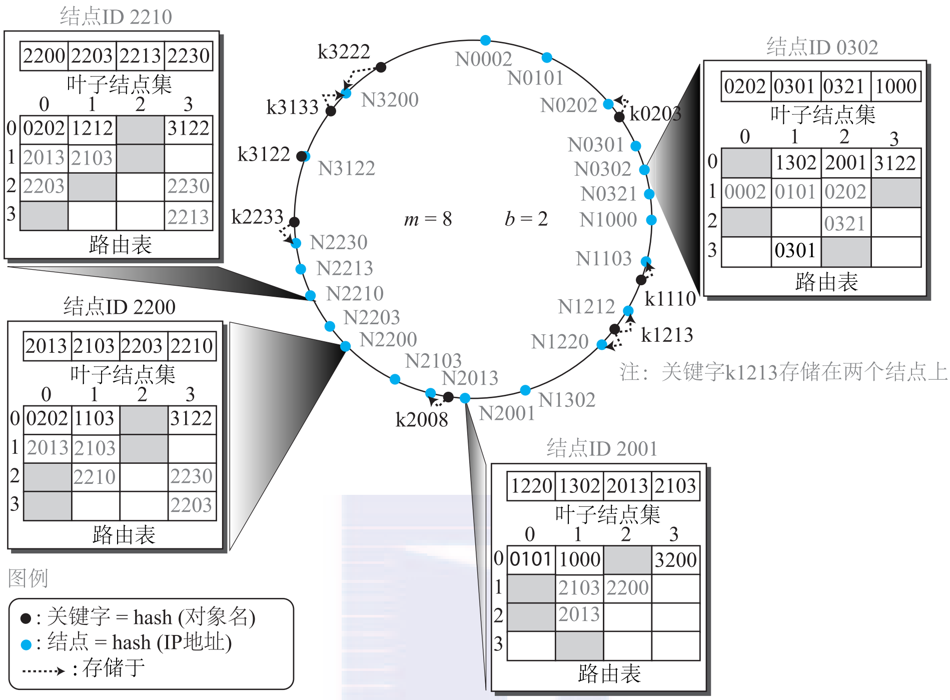


图 2-53　Pastry 环举例

查找

正如我们在 Chord 中讨论的，Pastry 中使用的一种操作是查找：给定一个关键字，我们需要找到存储着关键字相关信息或关键字自身的结点。表 2-20 以伪代码形式给出了查找操作。N 是本地结点的标识符，它是接收报文的结点同时也需要找到存储着报文中关键字的结点。

表 2-20　查找

```
Lookup (key)
{
    if (key is in the range of N's leaf set)
        forward the message to the closest node in the leaf set
    else
        route (key, Table)
}
route (key, Table)
{
    p = length of shared prefix between key and N
    v = value of the digit at position p of the key                // 从0开始的位置
    if (Table [p, v] exists)
        forward the message to the node in Table [p, v]
    else
        forward the message to a node sharing a prefix as long as the current node, but
        numerically closer to the key.
}
```

例 2.20　在图 2-53 中，我们假设结点 N2210 接收到一个查询，它寻找负责 key2008 的结点。因为当前结点不负责这个关键字，它首先检查自己的叶子结点集。key2008 不在叶子结点范围内，因此结点需要使用路由表。因为公共前缀的长度是 1，即 p = 1。关键字中第 1 位数字的值为 v = 0。结点在表[1，0]中检查标识符，得到结果 2013。查询被转发给结点 2013，它实际上负责这个关键字。结点将它的信息发送给请求结点。

例 2.21　在图 2-53 中，我们假设结点 N0302 接收到一个查询，它寻找负责 key0203 的结点。当前结点不负责这个关键字，但是这个关键字在它的叶子结点集范围内。这个集合中与 key 最近的结点是 N0202。查询被发送到这个结点，实际上就是负责这个关键字的结点。结点 N0202 将它的信息发送给请求结点

加入

加入 Pastry 中的环要比 Chord 中更简单更快速。一个新的结点 X 应该知道至少一个结点 N0，这个结点应该与 X 靠近（基于接近度）；这可以通过运行一个名为附近结点发现的算法来完成。结点 X 发送一个加入报文给结点 N0。在我们的讨论当中，我们假设 N0 的标识符与 X 的标识符没有公共前缀。以下步骤给出了结点 X 如何构造路由表及叶子结点集：

1. 结点 N0 发送第 0 行的内容给结点 X。因为两个结点没有公共前缀，结点 X 使用这个信息的适当部分来构造自己的第 0 行。之后，结点 N0 将加入报文作为查找报文进行处理，假设 X 的标识符为关键字。它向结点 N1 转发加入报文，N1 的标识符更接近 X。
2. 结点 N1 发送第 1 行的内容给结点 X。因为这两个结点有一个相同前缀，结点 X 使用这个信息的适当部分来构造自己的第 1 行。之后，结点 N1 将加入报文作为查找报文进行处理，假设 X 的标识符为关键字。它向结点 N2 转发加入报文，N2 的标识符更接近 X。
3. 这个过程继续，直到 X 的路由表完成。
4. 过程中的最后一个结点，与 X 有最长的公共前缀，它向结点 X 发送叶子结点集，这个集合变成了结点 X 的叶子结点集。
5. 然后结点 X 与它路由表中的结点交换信息和叶子结点集，从而来改善自己的路由信息并允许那些结点更新自身信息。

例 2.22　图 2-54 给出了标识符为 N2212 的新结点 X 是如何加入环的，结点 X 使用图 2-53 中四个结点的信息来创建初始路由表和叶子结点集。请注意，这两个表的内容在更新过程中将更接近它们的理想状态。在这个例子中，我们假设基于接近度，结点 0302 是结点 2212 的接近的结点。

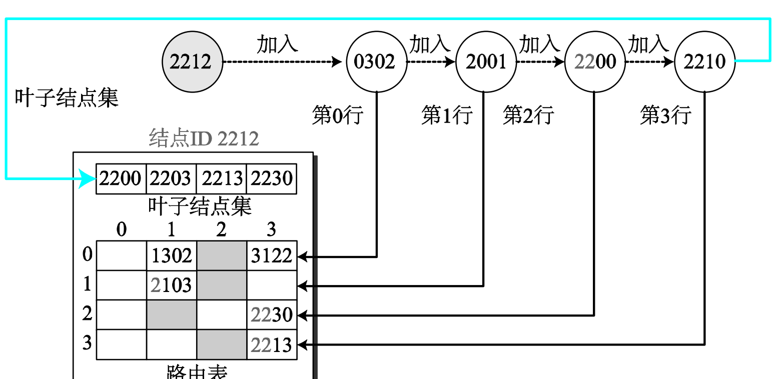


图 2-54　例 2.22

离开或失效

每个 Pastry 结点都将周期性地发送探测报文，来检测叶子结点集和路由表中的结点存活状况。

如果一个本地结点发现叶子结点集中的某个结点对探测报文无响应，它就假设这个结点已经失效或离开。为了替代这个结点，本地结点与叶子结点集中具有最大结点标识符的活动结点联系，并且利用那个结点的叶子结点集信息修复自己叶子结点集信息。由于靠近结点的叶子结点集有覆盖，因此这个过程是成功的。

如果本地结点发现路由表中[i，j]结点对探测报文没有响应，它就发送报文给同一行的活动结点并请求表[i，j]的标识符。这个标识符代替了失效或离开的结点。

应用

Pastry 被用于一些应用，比如 PAST，这是一个分布式文件系统，以及 SCRIBE，这是一个分散式发布/订阅系统。

2.4.5　Kademlia

另一种 DHT 对等网络是 **Kademlia**，它由 Maymounkov 和 Mazières 设计。与 Pastry 类似，Kademlia 基于结点距离来路由报文，但是正如下文所述，Kademlia 中的距离度量与 Pastry 中不同。在这个网络中，两个标识符（结点或关键字）之间的距离是通过位异或（XOR）来度量的。换言之，如果 x 和 y 是两个标识符，我们有

$$\text{distance}(x, y) = x \oplus y$$

当我们度量两点间的几何距离时，XOR 有以下四个特性：

$x \oplus x = 0$	点与自身的距离是 0。
$x \oplus y > 0$ if $x \neq y$	两点间距离大于 0。
$x \oplus y = y \oplus x$	x 到 y 的距离与 y 到 x 的距离相等。
$x \oplus z \leqslant x \oplus y + y \oplus z$	满足三角关系。

标识符空间

在 Kademlia 中，结点和数据项是 m 位标识符，它们创建了含有 2^m 个点的标识符空间，这些点分布在二叉树叶子结点上。此协议使用 SHA-1 散列算法，其中 $m = 160$。

例 2.23　为方便起见，我们假设 $m = 4$。在这个空间中有 16 个分布在二叉树叶子结点上的标识符。图 2-55 给出的情况只有 8 个活动结点以及 5 个关键字。

如图 2-55 所示，由于 $3 \oplus 3 = 0$，因此关键字 k3 存储在结点 N3 上。尽管关键字 k7 看起来与 N6 和 N8 在数字上等距，但是，它却存储在 N6 上，因为 $6 \oplus 7 = 1$ 而 $6 \oplus 8 = 14$。另一个有趣的点是关键字 k12，它与 N11 在数字上更接近，但它却存储在 N15 上，因为 $11 \oplus 12 = 7$ 但是 $15 \oplus 12 = 3$。

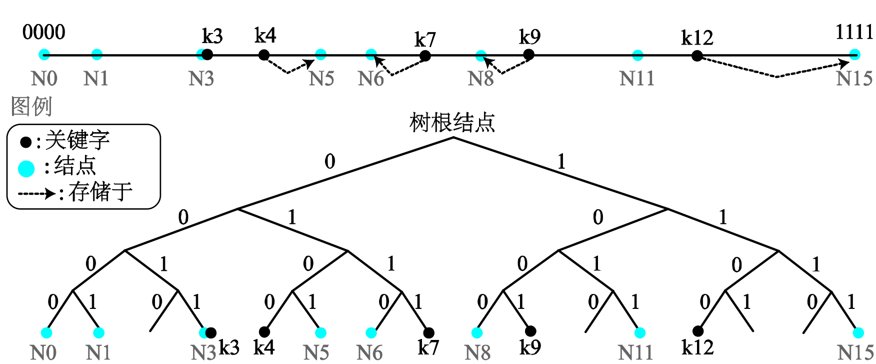


图 2-55　例 2.23

路由表

Kademlia 仅为每一个结点维护一个路由表；且没有叶子结点集。网络中每个结点将二叉树分成 m 棵子树，这些子树并不包含结点自身。子树 i 包含了那些与相应结点共享最左 i 位数字（公共前缀）

的结点。路由表有 m 行但只有 1 列。在我们的讨论中，我们假设每一行存储了相应子树中的一个结点的标识符，但是之后我们展示出 Kademlia 允许每行有多达 k 个结点。这个思想和 Pastry 相同，但是公共前缀的长度是基于比特位的个数，而不是基于 2^b 的数字个数。表 2-21 给出了路由表。

例 2.24　让我们来找例 2.23 中的路由表。为简便起见，我们假设每行仅使用一个标识符。因为 $m = 4$，每个结点有四个子树，这些子树对应路由表中的四行。每行的标识符代表在相应子树中与当前结点距离最近的点。图 2-56 给出所有的路由表，但是只有三棵子树。为了使图例更小，我们选择出 8 棵树。

表 2-21　Kademlia 中的一个结点的路由表

公共前缀长度	标　识　符
0	子树中公共前缀长度为 0 的最接近结点
1	子树中公共前缀长度为 1 的最接近结点
⋮	⋮
$m-1$	子树中公共前缀长度为 $m-1$ 的最接近结点

我们以结点 6 举例，使用相应的子树来解释一下如何构造路由表。对于其他结点的解释是类似的。

a. 在第 0 行，我们需要插入一个结点标识符，这个结点在子树中公共前缀长度 $p = 0$ 且距离最近。在这棵子树中有三个结点（N8、N11 以及 N15），然而，N15 与 N6 最近，因为 N6 ⊕ N8 = 14，N6 ⊕ N11 = 13 并且 N6 ⊕ N15 = 9。N15 就被插入到第 0 行。

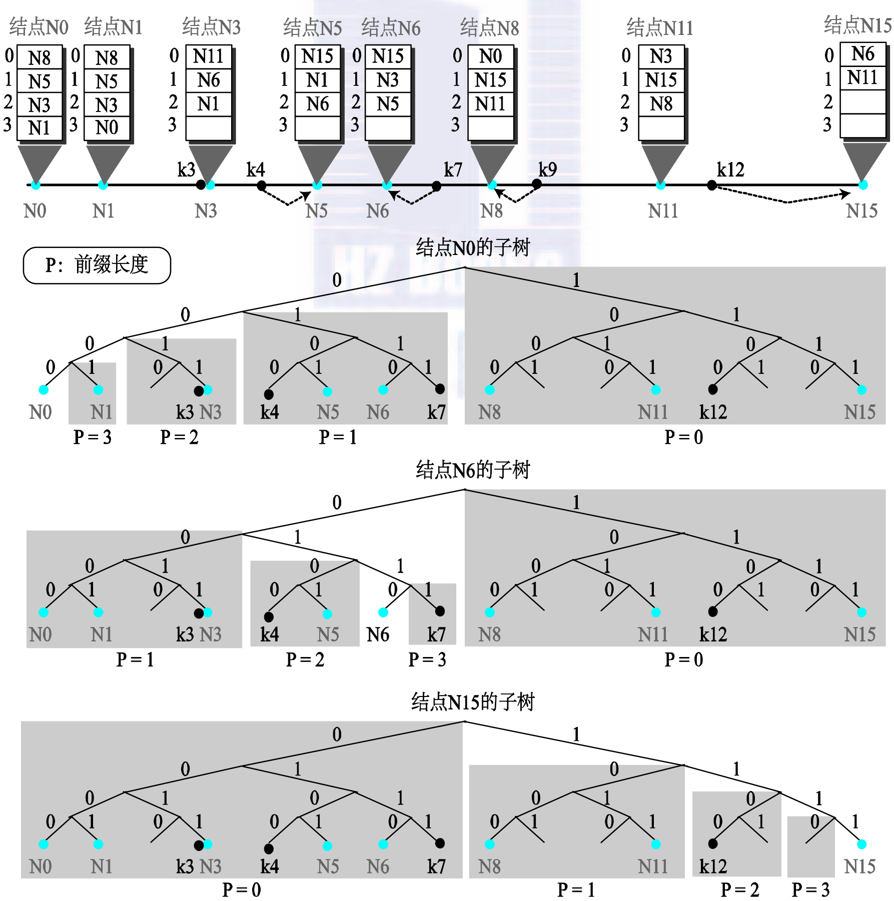


图 2-56　例 2.24

b. 在第 1 行，我们需要插入一个结点标识符，这个结点在子树中公共前缀长度 $p = 1$ 且距离

最近。在这棵子树中有三个结点（N0、N1 以及 N3），然而，N3 与 N6 最近，因为 N6 ⊕ N0 = 6，N6 ⊕ N1 = 7 并且 N6 ⊕ N3 = 5。N15 就被插入到第 1 行。

c. 在第 2 行，我们需要插入一个结点标识符，这个结点在子树中公共前缀长度 $p = 2$ 且距离最近。在这棵子树中只有一个结点（N5），它就被插入到那里。

d. 在第 3 行，我们需要插入一个结点标识符，这个结点在子树中公共前缀长度 $p = 3$ 且距离最近。在这棵子树中没有结点，因此这行为空。

例 2.25　在图 2-56 中，我们假设结点 N0（$0000)_2$ 接收到一个查找报文，报文要找负责 k12（$1100)_2$ 的结点。这两个标识符的公共前缀长度为 0。结点 N0 发送报文给路由表第 0 行的结点 N8。现在 N8（$1000)_2$ 需要查找与 k12（$1100)_2$ 最近的结点。这两个标识符的公共前缀长度为 1。结点 N8 将这个报文发送给路由表第 1 行的结点 N15，N15 负责这个关键字 k12。路由过程结束。这个路由是 N0→N8→N15。有趣的是结点 N15（$111)_2$ 以及 k12（$1100)_2$ 公共前缀长度为 2，但是 N15 的第 2 行是空的，这意味着 N15 它自身负责 k12。

例 2.26　在图 2-56 中，我们假设结点 N5（$0101)_2$ 接收到一个查找报文，报文要找负责 k7（$0111)_2$ 的结点。这两个标识符的公共前缀长度为 2。结点 N5 发送报文给路由表第 2 行的结点 N6，这个结点负责 k7。路由过程结束。这个路由是 N5→N6。

例 2.27　在图 2-56 中，我们假设结点 N11（$101)_2$ 接收到一个查找报文，报文要找负责 k4（$0100)_2$ 的结点。这两个标识符的公共前缀长度为 0。结点 N11 发送报文给路由表第 0 行的结点 N3。现在 N3（$0011)_2$ 需要查找与 k4（$0100)_2$ 最近的结点。这两个标识符的公共前缀长度为 1。结点 N3 将这个报文发送给路由表第 1 行的结点 N6。现在结点 N6（$0110)_2$ 需要查找与 k4（$0100)_2$ 最近的结点。这两个标识符的公共前缀长度是 2。结点 N6 发送报文给路由表 2 行的结点 N5，这个结点负责关键字 k12。路由过程结束。这个路由是 N11→N3→N6→N5。

k-桶

在之前的讨论中，我们假设路由表中每一行仅仅列出相应子树中的一个结点。为了提高效率，Kademlia 要求每一行至少维护 *k* 个来自相应子树的结点。*k* 的具体数值依赖于系统，但是实际网络中推荐使用 20 左右的数值。因此，路由表中每一行称为一个 k-桶（k-bucket）。在每一行中包含一个以上结点可以允许客户在结点离开网络或失效时使用替换结点。Kademlia 将那些在网络中连接很长时间的结点存储在桶中。已经证明的是，保持连接时间越长的结点越有可能在更长时间内保持连接。

并行查询

由于在 k-桶中有多个结点，Kademlia 允许向 k-桶顶部 α 个结点发送 α 个并行查询。如果一个结点失效或无法回应查询，并行查询将减少延迟。

并发更新

Kademlia 中另一个有趣的特性是并发更新。无论何时，当一个结点收到查询或响应，它都立即更新 k-桶。如果发向一个结点的多个查询没有收到响应，发送查询的结点将从相应的 k-桶中去除这个目的结点。

加入

如 Pastry 中，一个要加入网络的结点需要知道至少一个结点。加入的结点向网络中的结点发送自身标识符，好像这个标识符是要查找的关键字。它接收到的响应允许新结点创建自己的 k-桶。

离开或失效

当一个结点离开网络或失效时，其他结点使用如上所述的并发过程来更新它们的 k-桶。

2.4.6　一种流行的 P2P 网络：BitTorrent

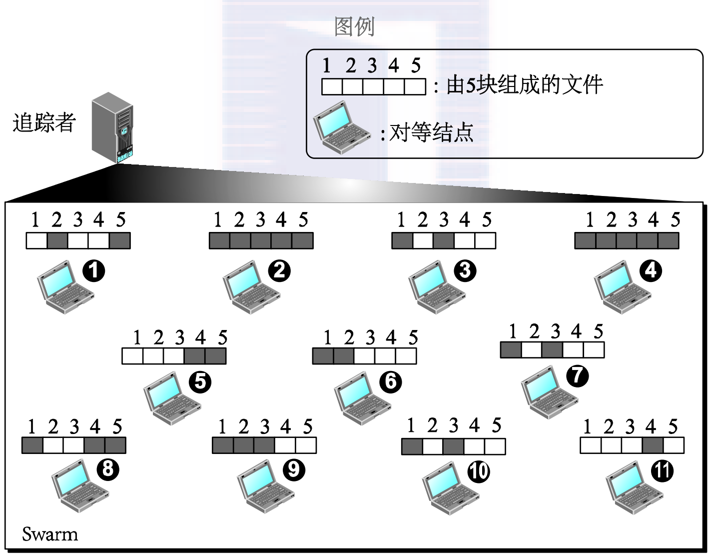
BitTorrent 是 Bram Cohen 设计的一个 P2P 协议，其目的是在一组对等结点间共享一个大文件。

然而，在本文中，共享（sharing）这个术语与其他文件共享协议中是不同的。此处，是一组对等结点参加到为组内所有对等结点提供文件拷贝的过程中，而不是一个对等结点允许另一个对等结点下载整个文件。文件共享是在合作过程中完成的，这称为 torrent。参加到 torrent 中的每个对等结点从另一个有文件的对等结点那里下载大文件的块，同时，它也为其他没有这个文件的结点上传文件块，它有点像孩子们玩的交易游戏以牙还牙（tit-for-tat）。参加 torrent 的所有对等结点的集合称为一个群（swarm）。群中拥有完整文件内容的对等结点称为种子（seed）；只有部分文件并想下载其余部分的对等结点称为寄生虫（leech）。换言之，一个群是种子和寄生虫的组合。BitTorrent 已有多个版本和实现。我们首先描述原始版本，它使用称为追踪者（tracker）的中心结点。之后，我们给出一些新的版本是如何利用 DHT 消除追踪者的。

带有追踪者的 BitTorrent

在原始 BitTorrent 中，在 torrent 中有另一个实体，叫做追踪者。正如其名所示，它追踪群的操作，后文会对其进行描述。图 2-57 给出了一个带有种子、寄生虫和追踪者的 torrent。

在图 2-57 中，文件要被共享，文件内容被分成五个片段（块）。对等结点 2 和 4 已经拥有所有文件片段；另外的对等结点有一些片段。每个对等结点拥有的文件片段标注为阴影。上传和下载文件片段将会继续。某些对等结点可能会离开 torrent；某些可能会加入。



注：对等结点2和4是种子；其他是寄生虫。

图 2-57 torrent 举例

现在假设一个新对等结点想要下载相同的文件内容。新对等结点访问 BitTorrent 服务器查找文件内容的名字。它接收到一个元文件，即 torrent 文件，这个文件包含了文件内容片段的信息以及处理这个 torrent 的追踪者的地址。现在新对等结点访问追踪者并取得 torrent 中一些对等结点的地址，通常称为邻居（neighbor）。现在新对等结点是 torrent 的一部分并且可以下载和上传文件内容的片段。当它拥有所有片段时，它可能会离开 torrent 或留在 torrent 中帮助其他对等结点得到文件内容的所有片段，这些被帮助的对等结点包括在其后加入的新结点。没有什么可以防止一个对等结点在得到所有片段之前就离开 torrent，而且在之后这个结点可以再加入或干脆不再加入 torrent。

尽管加入、共享、离开 torrent 的过程看似简单，BitTorrent 协议应用了一组策略来提供公平性，鼓励对等结点交换片段，防止某个对等结点接收请求过载并且允许对等结点寻找提供更好服务的结点。

为了避免过载并实现公平性，每个结点需要限制它与邻居们的并发连接；通常的数值为 4。一个对等结点将其邻居标记为非阻塞或阻塞。它也将它们标记为感兴趣或不感兴趣。

换言之，一个对等结点将他的邻居列表分为两个不同的组：*非阻塞*（unchoked）和*阻塞*（choked）。它也将其分为*感兴趣*（interested）和*不感兴趣*（uninterested）组。非阻塞组是那些被并发连接到的当前结点组；它从组中持续上传并下载片段。阻塞组是那些当前没有被连接到的结点组，但是将来可能连接。

每隔 10 秒，当前对等结点尝试连接组中感兴趣但是处于阻塞状态的对等结点，以获得更高的数据速率。如果这个新的对等结点比其他非阻塞对等结点拥有更高的速率，那么这个新对等结点可能变为非阻塞状态，并且非阻塞组中最低数据速率的结点可能移入阻塞组。采取这种方法，非阻塞组中的对等结点总是拥有被探测结点中最高的数据速率。使用这种策略，将邻居分成子组，其中那些有一致数据传输速率的邻居将相互通信。在这种策略中可以看到上文所述的以牙还牙交易策略思想。

为了允许一个尚无片段共享的新加入对等结点也能从其他结点接收片段，每隔 30 秒，一个对等结点随机地将一个感兴趣结点从阻塞组中选出来，标记为非阻塞，而不管其上传速率是多少。这个动作称为*乐观非阻塞*（optimistic unchocking）。

BitTorrent 协议采用一种称为*最少优先*（rarest-first）的策略，试图在每个对等结点每一刻拥有的不同片段数之间提供一种平衡。使用这种策略，对等结点首先试图下载邻居中重复得最少的片段。采用这种方法，这些稀少片段会传播得更快。

无追踪者的 BitTorrent

在 BitTorrent 原始的设计中，如果追踪者失效，新对等结点不能连接到网络且更新也中断。有几种 BitTorrent 实现消除了对中心化追踪者的需要。在此处描述的实现中，协议仍然使用追踪者，但不是一个中心追踪者。追踪的任务分布在网络中的一些结点上。在这一节，我们给出如何用 Kademlia DHT 实现这个目标，但是我们避免涉及特定的协议细节。

在带有中心追踪者的 BitTorrent 中，追踪者的任务是当给出原数据文件时，提供对等结点列表。如果我们将元数据的散列函数看做关键字，把群中对等结点列表的散列函数看做数值，我们可以使 P2P 网络中的某些结点起到追踪者的作用。一个加入 torrent 的新对等结点将元数据（关键字）的散列函数发送到它所知道的结点。P2P 网络使用 Kademlia 协议来寻找负责关键字的结点。负责结点向加入结点发送数值，这个数值实际上是对应 torrent 中的对等结点列表。现在加入结点可以使用 BitTorrent 协议与列表中的对等结点共享内容文件。

2.5 套接字接口编程

在 2.2 节，我们讨论了客户–服务器模式的原则。在 2.3 节，我们讨论了使用这种模式的一些标准应用。在这一节，我们给出如何使用过程编程语言 C 语言来编写简单的客户–服务器程序。我们选择 C 语言的原因有两个。第一，传统上说套接字编程就是从 C 语言开始的。第二，C 语言的底层特性将更好地揭示这类编程的精妙之处。在第 11 章，我们用 Java 扩展这种思想，它提供了一个更加简洁的版本。然而，即使跳过这一节也不会丧失在本书学习中的连续性。

C 的套接字接口

我们在 2.2 节讨论过套接字接口。在本节，我们给出这个接口用 C 语言是如何实现的。套接字接口的关键问题是理解套接字在通信中的角色。套接字没有存储待发送或待接收数据的缓冲区。它既不能发送也不能接收数据。套接字只起到一个引用或标签的作用。缓冲区和必要的变量在操作系统中创建。

套接字的数据结构

C 语言将套接字定义为一个结构（struct）。套接字结构由五个字段组成；每个套接字地址是一个由五部分构成的结构，如图 2-58 所示。请注意，程序员不该重定义这个结构；它已经在头文件中定义好了。我们简要讨论套接字结构中的五个字段。

- **族**。这个字段定义了协议簇（如何解释地址和端口号）。通常值是 PF_INET（用于当前因特网）、PF_INET6（用于下一代因特网）等等。我们在本节使用 PF_INET。
- **类型**。这个字段定义了四个套接字类型：SOCK_STREAM（用于 TCP）、SOCK_DGRAM（用于 UDP）、SOCK_SEQPACKET（用于 SCTP），以及 SOCK_RAW（用于直接使用 ISP 服务的应用）。
- **协议**。这个字段定义了族中特定协议。对于 TCP/IP 协议簇这个字段设置为 0，因为它是族中唯一的协议。
- **本地套接字地址**。这个字段定义了*本地套接字地址*。一个套接字地址是一个结构，它由长度字段、族字段（对于 TCP/IP 协议簇，它被设置为常量 AF_INET）、端口号字段（定义了进程）以及 IP 地址字段（定义了正在运行的进程所在的主机）构成。它也包含未使用字段。
- **远程套接字地址**。这个字段定义了远程套接字地址。它的结构与本地套接字地址相同。

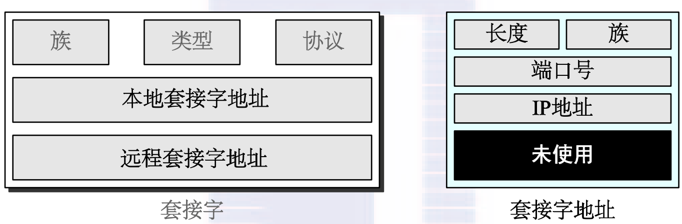


图 2-58　套接字数据结构

头文件

为了能够使用套接字的定义和所有在接口中定义的过程（函数），我们需要一组头文件。我们已经将所有这些头文件收集到了名为 headerFiles.h 的文件里。这个文件需要与程序创建到同一个文件夹中并且它的名字应该包含到所有程序中。

```
// "headerFiles.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>
#include <arpa/innet.h>
#include <sys/wait.h>
```

使用 UDP 迭代通信

正如我们之前讨论的，UDP 提供无连接服务器，其中客户发送请求，服务器返回响应。

用于 UDP 的套接字

在 UDP 通信中，客户和服务器每一端只使用一个套接字。服务器端创建的套接字永远运行；客户端创建的套接字在客户进程结束时被关闭（销毁）。图 2-59 给出了服务器和客户进程套接字的生存期。换言之，不同的客户使用不同的套接字，但是服务器只创建一个套接字，并且每次当一个新客户

建立连接时只改变远程套接字地址。这是符合逻辑的，因为服务器确实知道自身的套接字地址，但是不知道需要服务的客户端的套接字地址；在填充套接字的这一项之前，它需要等待客户进行连接。

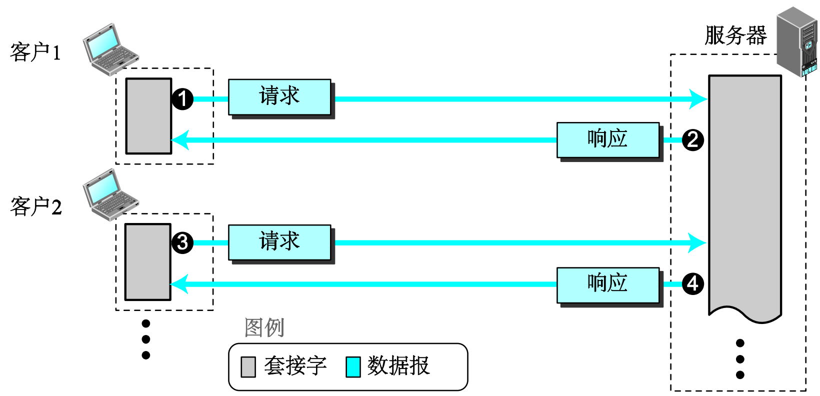


图 2-59　UDP 通信套接字

通信流程图

图 2-60 给出了一个简单的迭代通信的流程图。图中有多个客户，但是只有一个服务器。每个客户在每次循环中得到服务。请注意，这里没有连接建立以及连接终止。每个客户发送一个数据报并接收一个数据报。换言之，如果一个客户想要发送两个数据报，它就被认为是两个客户。第二个数据报需要等待循环轮次。

服务器进程　服务器进行被动开启（passive open），在被动开启中它做好连接准备，但是等待客户进程创建连接。它调用 socket 函数去创建套接字。在这个程序调用中的参数填充了前三个字段，但是本地和远程套接字地址字段仍然未定义。之后，服务器进程调用 bind 函数来填充本地套接字地址字段（信息来自操作系统）。然后，它调用另一个称为 recvform 的函数。然而这个函数阻塞服务器进程直到一个客户数据报到达。当一个数据报到达时，服务器进程解除阻塞并且从数据报中抽出数据报。它也抽取发送套接字地址用来在下一步中使用。在请求被处理之后，响应就准备好了，服务器进程将接收报文中的发送套接字地址填充到远程套接字地址，这样就完成了套接字结构。现在准备发送数据报。这通过调用另一个称为 sendto 的函数来完成。请注意，在服务器进程发送响应之前，套接字中的所有字段都应该被填充。在发送响应之后，服务器进程开始一个新的迭代并且等待其他客户连接。远程套接字地址字段将被再次填充上一个新的客户地址（或是同一个客户，而此处认为是新客户）。服务器进程是一个无限的进程；它永远运行。服务器套接字从不关闭，除非出现了问题且进程需要终止。

客户进程　客户进程进行主动开启（active open）。换言之，它开启连接。它调用 socket 函数来创建一个套接字并填充前三个字段。尽管某些实现要求客户进程也调用 bind 函数来填充本地套接字，但通常这是由操作系统自动完成的，操作系统为客户选择一个临时端口号。之后，客户进程调用 sendto 函数，并提供远程套接字地址信息。这个套接字地址必须由客户进程的用户提供。套接字在这个时刻完成并且数据报被发送。现在客户进程调用 recvfrom 函数，它阻塞了客户进程，直到响应来自服务器进程。此处没有必要从这个函数中提取远程套接字地址，因为此处没有调用 sendto 函数。换言之，服务器端和客户端的 recvfrom 函数行为不同。在服务器进程中，recvfrom 函数首先被调用，然后调用 sendto，因此 sendto 所使用的远程套接字地址可以从 recvfrom 获得。在客户进程，sendto 在 recvfrom 之前被调用，因此远程地址应该由程序使用者使用，使用者知道她想连接的是哪一台服务器。最终 close 函数被调用以销毁套接字。注意，客户进程是有限的；在响应

被接收之后，客户进程终止。尽管我们可以使用一个循环来设计发送多个数据报的客户，但是每次循环迭代对于服务器都像一个新的客户。

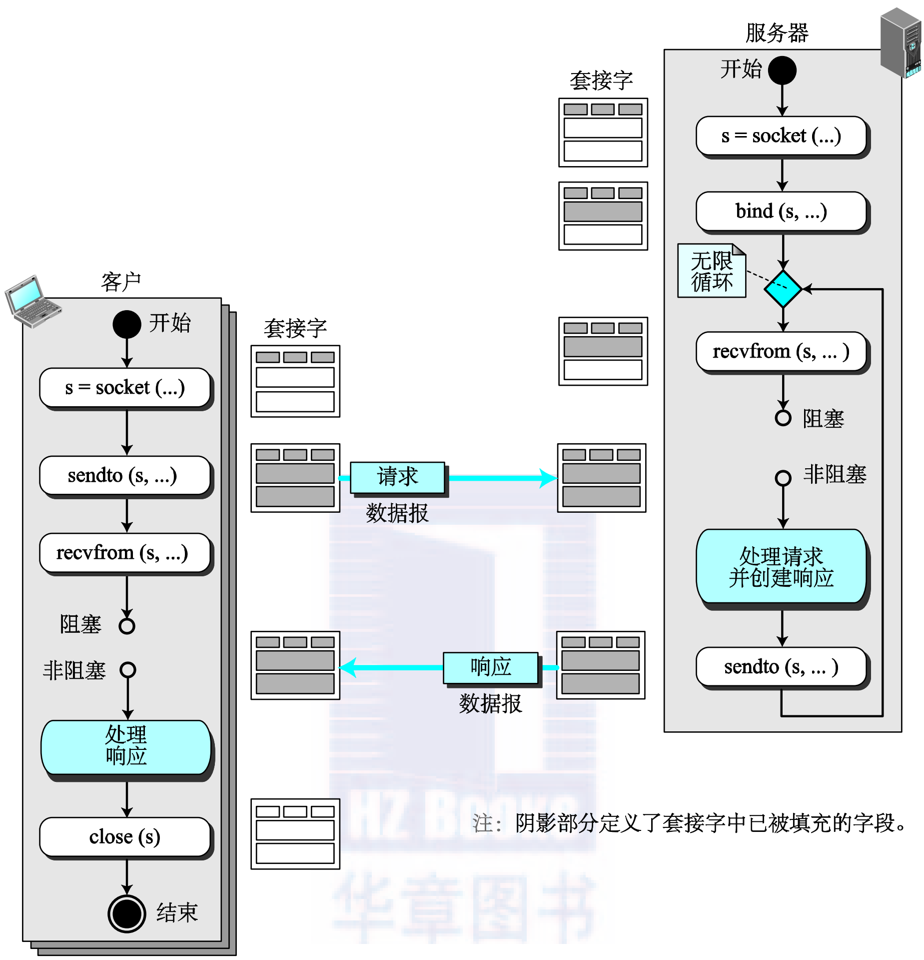


图 2-60　迭代 UDP 通信的流程图

编程举例

在这一节，我们给出如何进行客户和服务器编程来模拟使用 UDP 的标准回送（echo）应用。客户程序向服务器发送一个短的字符串；服务器回送检验这个字符串。标准应用供客户计算机使用，来检测另一台服务器计算机是否活动。我们的程序比标准使用中的程序更简单；为简单起见，我们省略了一些错误检测和调试细节。

回送服务器程序　表 2-22 给出了使用 UDP 的回送服务器编程。程序依照图 2-60 中的流程图。

表 2-22　使用 UDP 的回送服务器程序

```
// UDP回送服务器程序
#include "headerFiles.h"
int main (void)
{
    // 声明以及定义变量
    int s;                                    // 套接字描述符（引用）
```

```
7       int len;                                              // 要回送的字符串的长度
8       char   buffer [256];                                  // 数据缓冲区
9       struct sockaddr_in servAddr;                          //服务器（本地）套接字地址
10      struct sockaddr_in clntAddr;                          // 客户（远程）套接字地址
11      int  clntAddrLen;                                     //客户套接字地址的长度
12      //建立本地（服务器）套接字地址
13      memset (&servAddr, 0, sizeof (servAddr));                          // 分配内存
14      servAddr.sin_family =   AF_INET;                                   // 族字段
15      servAddr.sin_port = htons (SERVER_PORT)                            // 默认端口号
16      servAddr.sin_addr.s_addr =  htonl (INADDR_ANY);                    // 默认IP地址
17      // 创建套接字
18      if ((s = socket (PF_INET, SOCK_DGRAM, 0) < 0);
19      {
20              perror ("Error: socket failed!");
21              exit (1);
22      }
23      // 将套接字绑定到本地地址和端口
24      if ((bind (s, (struct sockaddr*) &servAddr, sizeof (servAddr)) < 0);
25      {
26              perror ("Error: bind failed!");
27              exit (1);
28      }
29      for ( ; ; )         // 永远运行
30      {
31              // 获取字符串
32              len = recvfrom (s, buffer, sizeof (buffer), 0,
33                           (struct sockaddr*)&clntAddr, &clntAddrLen);
34              // 发送字符串
35              sendto (s, buffer, len, 0, (struct sockaddr*)&clntAddr, sizeof(clntAddr));
36      } // 循环结束
37   } // 回送服务器程序结束
```

第 6 行到第 11 行声明并定义了程序中使用的变量。第 13 行到第 16 行为服务器套接字地址分配内存（使用 memset 函数）并且使用传输层提供的默认值填充了套接字地址字段。为了插入端口号，我们使用 htons（host to network short）函数，它将主机字节序格式的短整型数值转换为网络字节序格式。为了插入 IP 地址，我们使用 htonl（host to network long）函数来做相同的事情。

第 18 行到第 22 行在 if 声明中调用 socket 函数来检查错误。因为如果调用失败，这个函数就返回–1，程序打印错误消息并且退出。perror 函数是 C 中的标准错误函数。类似地，第 24 行到第 28 行调用 bind 函数来将套接字绑定到服务器套接字地址。这个函数也在 if 声明中被调用，从而检查错误。

第 29 行到第 36 行使用无限循环在每次迭代中为客户提供服务。第 32 到第 33 行调用 recvfrom 函数读取客户发送的请求。注意，这个函数是阻塞函数；当它消除阻塞时，它接收请求报文，同时提供客户套接字地址来完成套接字的最后部分。第 35 行调用 sendto 函数来给客户发回（回送）相同的报文，它使用 recvfrom 报文中获得的客户套接字地址。注意这里没有对请求报文进行加工；服务器仅回送它所收到的内容。

回送客户程序　表 2-23 给出了使用 UDP 的回送客户程序。程序依照图 2-60 中的流程图。

表 2-23　使用 UDP 的回送客户程序

```
// UDP 回送客户程序
#include "headerFiles.h"
int main (int argc, char* argv[ ])                //三个参数之后待检验
{
      // 声明并定义变量
      int   s;                                    // 套接字描述符
      int   len;                                  // 要回送的字符串的长度
      char*  servName;                            // 服务器名
      int  servPort;                              // 服务器端口
      char*  string;                              // 要回送的字符串
      char  buffer [256 + 1];                     // 数据缓冲区
      struct sockaddr_in  servAddr;               // 服务器套接字地址
      //检测并设置程序参数
      if (argc != 3)
      {
            printf ("Error: three arguments are needed!");
            exit(1);
      }
      servName = argv[1];
      servPort = atoi (argv[2]);
      string  =  argv[3];
      //建立服务器套接字地址
      memset (&servAddr, 0, sizeof (servAddr));
      servAddr.sin_family = AF_INET;
      inet_pton (AF_INET, servName, &servAddr.sin_addr);
      servAddr.sin_port = htons (servPort);
      //创建套接字
      if ((s = socket (PF_INET, SOCK_DGRAM, 0) < 0);
      {
            perror ("Error: Socket failed!");
            exit (1);
      }
      //发送回送字符串
      len =  sendto (s, string, strlen (string), 0, (struct sockaddr)&servAddr, sizeof (servAddr));
      //接收回送字符串
      recvfrom (s, buffer, len, 0, NULL, NULL);
      //打印并验证回送字符串
      buffer [len] = '\0';
      printf ("Echo string received: ";
      fputs (buffer, stdout);
      //关闭套接字
      close (s);
      //停止程序
      exit (0);
} //回送客户程序结束
```

第 6 行到第 12 行声明并定义了程序中使用的变量。第 14 行到第 21 行检测并设置了程序运行时提供的参数。头两个参数提供了服务器名与服务器端口号；第三个参数是被回送的字符串。第 23 到第 26 行分配内存、调用函数 inet_pton 将服务器名转换成服务器 IP 地址、将端口号转换为适当的字节序。inet_pton 函数调用 DNS（在本章前面讨论过）。这三部分信息是 sendto 函数所需要的，都被存储在适当的变量中。

第 34 行调用 sendto 函数发送请求。第 36 行调用 recvfrom 函数接收回送报文。请注意报文中的两个参数是 NULL，因为我们不需要取出远程站点的套接字地址；报文已经被发送了。

第 38 行到第 40 行用来在屏幕上显示回送报文，从而达到调试的目的。注意，在第 38 行我们在回送报文尾部加入了一个空字符，使得其他内容可以在下一行显示。最终，第 42 行关闭套接字，第 44 行离开程序。

使用 TCP 通信

正如我们之前描述的，TCP 是面向连接的协议。在发送或接收数据之前，需要在客户端和服务器之间建立连接。在连接建立之后，只要它们有数据要发送或接收，两端就可以彼此发送以及接收数据块。TCP 连接可以是迭代的（一次服务一个客户）也可以是并发的（一次服务多个客户）。在本节，我们只讨论迭代方法。并发方法参见 Java 部分（见第 11 章）。

TCP 中使用的套接字

TCP 服务器使用两个不同的套接字，一个用于连接建立，一个用于数据传输。我们把第一个称为监听套接字（listen socket）并把第二个称为套接字（socket）。设置两种套接字的目的是将建立阶段和数据交换阶段分开。服务器使用监听套接字来监听试图建立连接的新客户。在连接建立之后，服务器创建一个用于和客户交换数据的套接字并且最终终止连接。客户只使用一个套接字用于连接建立以及数据交换（见图 2-61）。

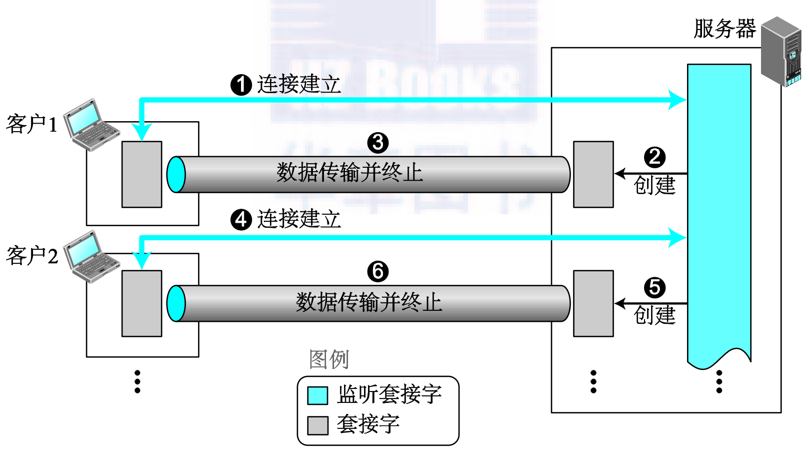


图 2-61　TCP 通信中使用的套接字

通信流程图

图 2-62 给出了简化的迭代通信流程图。图中有多个客户但是只有一个服务器。每个客户在每次循环中被服务。这幅流程图与 UDP 中的流程图类似，但是有一些不同，我们会对每一端进行解释。

服务器进程　在图 2-62 中，和 UDP 服务器进程一样，TCP 服务器进程调用 socket 和 bind 函数，但是这两个函数创建监听套接字，它只在连接建立阶段被使用。之后，服务器进程调用 listen 函数，允许操作系统开始接收客户、完成连接阶段并把他们放入等待被服务的列表。这个函数也定义了被连接的客户等待列表的大小，这依赖于服务器进程的复杂性，但是通常值为 5。

现在，服务器进程开始循环并且逐一对客户进行服务。在每次循环中，服务器进程调用 accept

函数从已连接客户的等待列表中去除一个客户，对其进行服务。如果列表是空的，那么 accept 函数进入阻塞状态直到出现一个客户待服务。当 accept 函数返回，它创建一个新的与监听套接字一样的套接字。监听套接字现在移入后台，并且新的套接字成为活动套接字。服务器进程现在使用连接建立期间获得的客户套接字地址，用它来填充新建套接字的远程套接字地址。

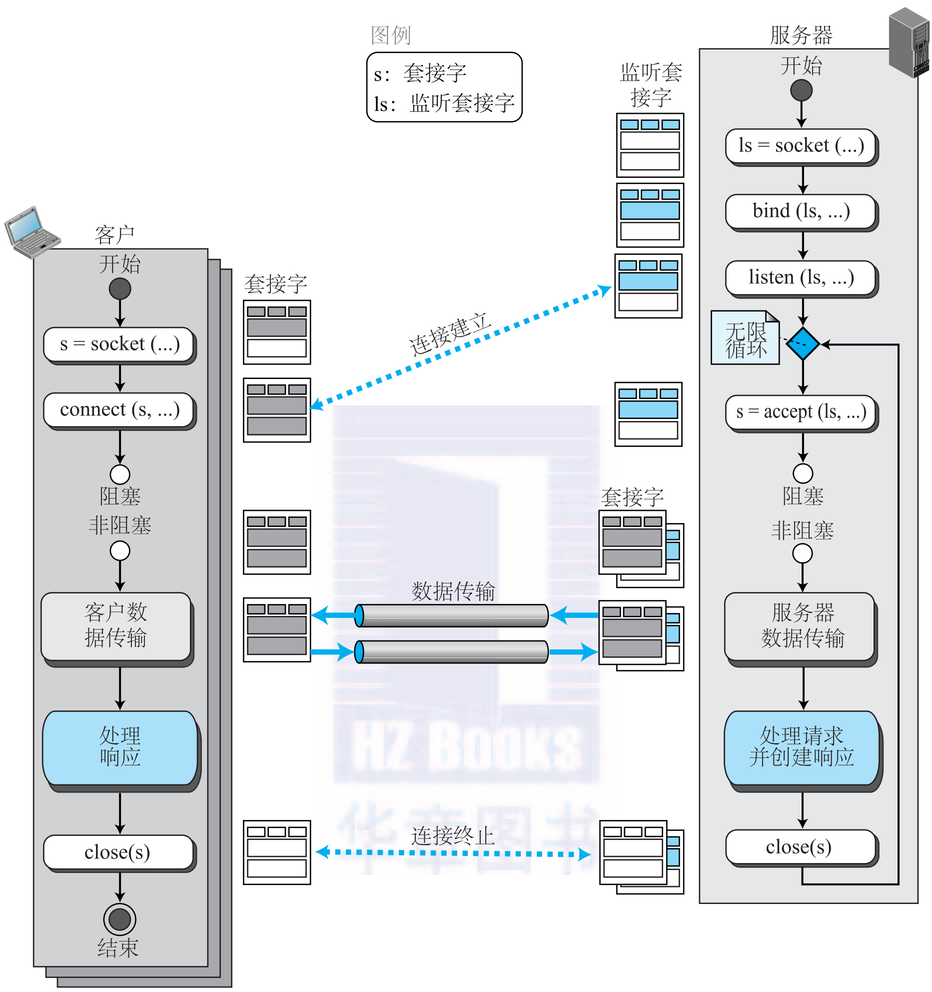


图 2-62　迭代 TCP 通信流程图

此时，客户和服务器可以交换数据。我们没有给出数据传输的特定方式，因为这取决于特定的客户–服务器对。TCP 使用 send 以及 recv 程序在它们之间传输数据字节。这两个函数比 UDP 中使用的 sendto 和 recvfrom 函数更简单，因为它们不提供远程套接字地址；连接已经在客户和服务器之间建立。然而，由于 TCP 用于传输无边界报文，每个应用需要仔细设计数据传输部分。send 和 recv 函数可能被调用多次来处理大量数据传输。可以将图 2-62 中的流程图当作一个通用流程图；如果是特殊用途，需要定义服务器数据传输（sever data-transfer）盒。当我们讨论回送客户–服务器程序时，我们将这样做一个简单的例子。

客户进程　客户流程图与 UDP 版本类似，除了*客户数据传输*（client data-transfer）盒需要为每个特定情况定义。当稍后我们编写特定程序时，我们会这么做。

编程举例

在这一节，我们给出如何编写客户和服务器程序来模拟使用 TCP 的标准回送应用。客户程序

发送一个短的字符串给服务器；服务器将相同的字符串回送到客户。然而，在我们这样做之前，我们需要为客户和服务器数据传输盒提供流程图，如图 2-63 所示。

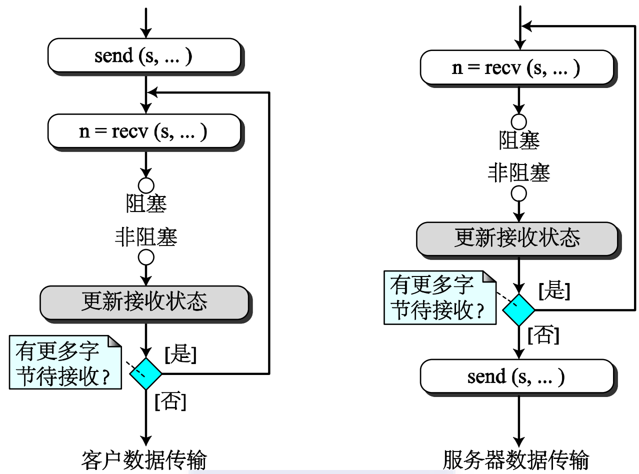


图 2-63　客户和服务器数据传输盒的流程图

对于这个特定的情况，因为待发送的字符串很短（小于几个单词），我们可以在客户端调用 send 函数一次完成。然而，TCP 并不保证把整个报文在一个报文段内发送。因此，我们需要在服务器端调用一组 recv（在一个循环内）来接收整个报文并将它们收集到缓冲区内，从而能一次性发送回去。当服务器向客户发送回送报文时，它也可能使用多个报文段，这意味着客户的 recv 程序需要调用多少次就会被调用多少次。

另一个有待解决的问题是设置缓冲区，缓冲区用于在每个站点接收数据。我们需要控制接收的字节数以及下一个数据块存储的位置。如图 2-64 所示，程序设置了一些变量进行控制。在每次迭代中，指针（ptr）移动指向下一个要接收的字节，接收字节的长度（len）呈增长趋势并且待接收的最大字节数（maxLen）呈减少趋势。

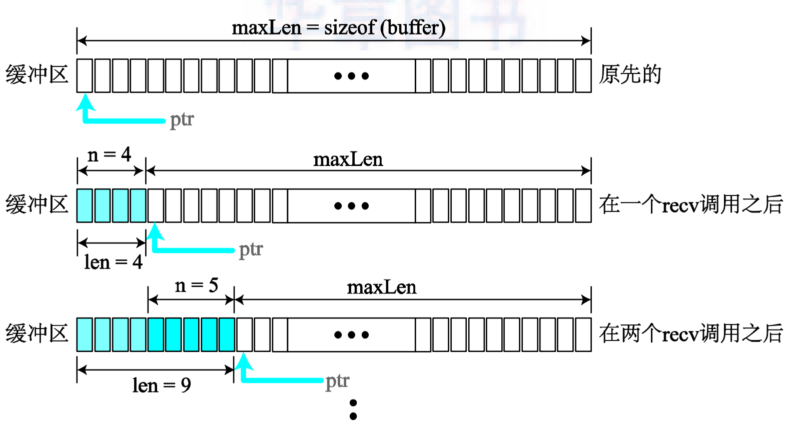


图 2-64　用于接收的缓冲区

考虑到以上两点之后，现在我们编写服务器和客户程序。

回送服务器程序　表 2-24 给出了使用 TCP 的回送服务器程序。程序遵循图 2-62 中的流程图。在图中，阴影中的每一部分对应于一条指令。带斜线部分显示了数据传输部分要做的处理工作。

第 6 行到第 16 行声明并定义了变量。第 18 行到第 21 行分配内存并且按 UDP 情况下所述创建了

本地（服务器）套接字地址。第 23 行到第 27 行创建了监听套接字。第 29 行到第 33 行将监听套接字绑定到第 18 行到第 21 行创建的服务器套接字地址上。第 35 行到第 39 行是 TCP 通信中的新内容。

调用 listen 函数让操作系统完成连接建立阶段并将客户置入等待列表。第 44 行到第 48 行调用 accept 函数来移除等待列表中的第一个客户并开始为其服务。如果在等待列表中没有客户，那么这个函数处于阻塞状态。第 50 行到第 56 行对图 2-63 中描述的数据传输部分进行编码。最大缓冲区大小与回送字符串长度都和图 2-64 中所示相同。

回送客户程序　表 2-25 给出使用 TCP 的回送客户程序。这个程序遵循图 2-62 的流程图。在图中，阴影中的每一部分对应于一条指令。带斜线部分显示了数据传输部分要做的处理工作。

表 2-24　使用 TCP 服务的回送服务器程序

```
1  // 回送服务器程序
2  #include "headerFiles.h"
3  int main (void)
4  {
5      // 声明并定义
6      int ls;                                    // 监听套接字描述符（引用）
7      int s;                                     // 套接字描述符（引用）
8      char buffer [256];                         // 数据缓冲区
9      char* ptr = buffer;                        // 数据缓冲区
10     int len = 0;                               // 待发送或接收的字节数
11     int maxLen = sizeof (buffer);              // 最大接收字节数
12     int n = 0;                                 // 每次调用recv接收的字节数
13     int waitSize = 16;                         // 等待客户数量
14     struct sockaddr_in serverAddr;             // 服务器地址
15     struct sockaddr_in clientAddr;             //  客户地址
16     int clntAddrLen;                           // 客户地址长度
17     // 创建本地（服务器）套接字地址
18     memset (&servAddr, 0, sizeof (servAddr));
19     servAddr.sin_family = AF_INET;
20     servAddr.sin_addr.s_addr = htonl (INADDR_ANY);     // 默认IP地址
21     servAddr.sin_port = htons (SERV_PORT);             // 默认端口
22     // 回创建监听套接字
23     if (ls = socket (PF_INET, SOCK_STREAM, 0) < 0);
24     {
25         perror ("Error: Listen socket failed!");
26         exit (1);
27     }
28     //  将套接字绑定到本地套接字地址
29     if (bind (ls, &servAddr, sizeof (servAddr)) < 0);
30     {
31         perror ("Error: binding failed!");
32         exit (1);
33     }
34     // 监听连接请求
35     if (listen (ls, waitSize) < 0);
36     {
37         perror ("Error: listening failed!");
```

```
            exit (1);
        }
        // 处理连接
        for ( ; ; )                        // 永远运行
        {
            // 接收来自客户的连接
            if (s = accept (ls, &clntAddr, &clntAddrLen) < 0);
            {
                perror ("Error: accepting failed!);
                exit (1);
            }
            // 数据传输部分
            while ((n = recv (s, ptr, maxLen, 0)) > 0)
            {
                ptr + = n;                 // 在缓冲区上移动指针
                maxLen - = n;              // 调整待接收的最大字节数
                len + = n;                 // 更新已接收的字节数
            }
            send (s, buffer, len, 0);      // 发回（回送）所有接收的字节
            // 关闭套接字
            close (s);
        } //循环结束
} // 回送服务器程序结束
```

表 2-25　回送客户程序

```
// TCP回送客户程序
#include "headerFiles.h"
int main (int argc, char* argv[ ])                // 三个参数之后待检验
{
    // 声明并定义
    int  s;                                       // 套接字描述符
    int  n;                                       // 每次调用recv接收的字节数
    char* servName;                               //服务器名
    int servPort;                                 //服务器端口号
    char* string;                                 //被回送的字符串
    int  len;                                     // 被回送的字符串的长度
    char  buffer [256 + 1];                       //缓冲区
    char* ptr = buffer;                           //在缓冲区上移动指针
    struct sockaddr_in serverAddr;                // 服务器套接字地址
    //检测并设置参数
    if (argc != 3)
    {
        printf ("Error: three arguments are needed!");
        exit (1);
    }
    servName = arg [1];
    servPort = atoi (arg [2]);
```

```
23        string = arg [3];
24        // 创建远程（服务器）套接字地址
25        memset (&servAddr, 0, sizeof(servAddr);
26        serverAddr.sin_family = AF_INET;
27        inet_pton (AF_INET, servName, &serverAddr.sin_addr); // 服务器IP地址
28        serverAddr.sin_port = htons (servPort);              // 服务器端口号
29        // 创建套接字
30        if ((s = socket (PF_INET, SOCK_STREAM, 0) < 0);
31        {
32            perror ("Error: socket creation failed!");
33            exit (1);
34        }
35        // 连接到服务器
36        if (connect (sd, (struct sockaddr*)&servAddr, sizeof(servAddr)) < 0);
37        {
38            perror ("Error: connection failed!");
39            exit (1);
40        }
41        // 数据传输部分
42        send (s, string, strlen(string), 0);
43        while ((n = recv (s, ptr, maxLen, 0)) > 0)
44        {
45            ptr + = n;                        // 在缓冲区上移动指针
46            maxLen - = n;                     // 调整待接收的最大字节数
47            len += n;                         // 更新已接收的字节数
48        } // while循环结束
49        // 打印并验证回送的字符串
50        buffer [len] = '\0';
51        printf ("Echoed string received: ");
52        fputs (buffer, stdout);
53        // 关闭套接字
54        close (s);
55        // 停止程序
56        exit (0);
57    } // 回送客户程序结束
```

TCP 的客户程序与 UDP 的客户程序非常相似，只有些许不同。因为 TCP 是面向连接的协议，第 36 行到第 40 行调用 connect 函数连接服务器。第 42 行到第 48 行使用图 2-63 中的思想完成数据传输。按图 2-64 所示方式完成接收数据的长度调整和指针移动。

2.6 章末资料

推荐读物

想要得到本章讨论主题的更多细节，我们推荐如下书籍和 RFC。在本书末列出了方括号中的参考资料。

书籍

此处列出一些涵盖本章内容的书籍，它们包括[Com 06]、[Mir 07]、[Ste 94]、[Tan 03]和[Bar et al. 05]。

RFC

HTTP 在 RFC2068 和 2109 中讨论到。FTP 在 RFC 959、 2577 和 2585 中讨论。TELNET 在 RFC 854、855、856、1041、1091、1372 和 1572 中讨论。SSH 在 RFC 4250、4251、4252、4253、4254 和 4344 中讨论。DNS 在 RFC1034、1035、1996、2535、3008、3658、3755、3757、3845、3396 以及 3342 中讨论。SMTP 在 RFC2821 和 2822 中讨论。POP3 在 RFC 1939 中有解释。MIME 在 RFC 2046、2047、2048 和 2049 中讨论。

小结

因特网中的应用或使用客户–服务器模式或使用对等模式。在客户–服务器模式中，一个应用程序称为服务器，它提供服务，另一个应用程序称为客户，它接受服务。服务器程序是一个无限程序；客户程序是有限的。在对等模式中，一个对等结点既可以是一个客户也可以是一个服务器。

万维网（WWW）是信息宝库，它把全世界的结点连接在一起。超文本和超媒体文档通过指针互相连接。超文本传输协议（HTTP）是万维网（WWW）上用于访问数据的主要协议。

文件传输协议（FTP）是一个 TCP/IP 客户–服务器应用，它用于从一个结点向另一个结点拷贝件。FTP 要求为数据传输提供两个连接：一个控制连接和一个数据连接。在非近似系统的之间的通信中 FTP 使用 NVT ASCII。

电子邮件是因特网最常见的应用。电子邮件体系结构包含几个部分，比如用户代理（UA）、报文传输代理（MTA）以及报文访问代理（MAA）。实现 MTA 的协议称为简单邮件协议（SMTP）。有两个协议用于实现 MAA：邮局协议版本 3（POP3）和因特网邮件访问协议 4（IMAP4）。

TELNET 是客户–服务器应用，它允许用户登录一台远程计算机，并使得用户能够访问远程系统。当用户通过 TELNET 进程访问远程系统时，这相当于分时环境。

域名系统（DNS）是一个客户–服务器应用，它用唯一的名称标识因特网中的主机。DNS 以层次结构组织名字空间，以此来将命名中所涉及的责任分散。

在对等网络中，准备好分享它们资源的因特网用户成为了对等结点并形成一个网络。对等网络分为中心化和分布式两种。在中心化 P2P 网络中，目录系统使用客户–服务器模式，但是文件存储和下载是通过对等模式完成的。在分布式网络中，目录系统以及文件的存储和下载都通过对等模式完成。

2.7 习题集

测试题

本章的交互式测试题请参见这本书的网站。在进行其他练习之前，强烈建议学生完成这些测试题以检查对这些内容的理解程度。

练习题

Q2-1 假设我们加入一个新的协议到应用层上。我们需要对其他层做什么改动？

Q2-2 请解释在客户–服务器模式中哪个实体提供服务，哪个实体接收服务？

Q2-3 在客户–服务器模式中，请解释为什么服务器应该一直运行而客户可以在需要时运行。

Q2-4 可否在一个只安装了 TCP 作为传输层协议的计算机上编写一个使用 UDP 服务的程序？请解释。

Q2-5 在周末，Alice 经常要从她家的笔记本电脑访问存储在办公室桌面的文件。上个星期，她在办公室桌面上安装了 FTP 服务器进程并在家里的笔记本上安装了 FTP 客户进程。当她周末无法访问文件时，感到很失望。哪里错了吗？

Q2-6 安装在个人计算机上的绝大多数操作系统有很多客户进程，但通常没有服务进程。请解释原因。

Q2-7 要设计一个使用客户–服务器模式的新应用。如果在客户和服务器之间只需要交换小报文而不用担心报文丢失或出错，你推荐使用什么传输层协议？

Q2-8 以下哪项可以是数据源？

a．键盘　　b．显示器　　c．套接字

Q2-9　源套接字地址是 IP 地址和端口号的组合。请解释每部分定义的内容。

Q2-10　请解释客户进程如何得到要插入远程套接字地址中的 IP 地址和端口号。

Q2-11　如果一个 HTTP 请求需要在服务器端运行一个程序并且将结果下载到客户服务器，这个程序是以下的

a．静态文档　　b．动态文档　　c．活动文档

Q2-12　假设我们设计了一个新的客户–服务器应用程序，它需要持续连接。我们能使用 UDP 作为传输层协议吗？

Q2-13　Alice 有一个视频片段，Bob 想得到它；Bob 有另一个视频片段，Alice 想得到它。Bob 创建了一个网页并在 HTTP 服务器上运行。Alice 如何得到 Bob 的视频片段？Bob 能够得到 Alice 的吗？

Q2-14　当 HTTP 服务器接收到来自 HTTP 客户的请求报文时，服务器如何知道所有头部以及跟随其后的报文主体何时到达？

Q2-15　在非持续 HTTP 连接中，HTTP 如何通知 TCP 协议报文已到达结尾？

Q2-16　在日常生活通信中，你能找到与 FTP 的控制和数据连接相似的两个分离的连接吗？

Q2-17　FTP 使用两个不同的熟知端口号用于控制和数据传输。这是否意味着为了交换控制信息和数据而创建了两个分离的 TCP 连接？

Q2-18　FTP 使用 TCP 服务交换控制信息并进行数据传输。FTP 能够为这两种连接使用 UDP 服务吗？请解释。

Q2-19　在 FTP 中，哪个实体（客户或服务器）开启（主动开启）控制连接？哪个实体开启（主动开启）数据传输连接？

Q2-20　如果控制连接在 FTP 会话结束前被中断，你认为会发生什么？它会影响数据连接吗？

Q2-21　在 FTP 中，如果客户需要从服务器站点获取一个文件并且存储一个文件到服务器站点上，这需要多少个控制连接和数据传输连接？

Q2-22　在 FTP 中，服务器可以从客户端获取文件吗？

Q2-23　在 FTP 中，服务器能够从客户端得到文件列表或目录吗？

Q2-24　FTP 可以在两个使用不同操作系统的主机之间传输文件。这是什么原因？

Q2-25　在控制连接期间，FTP 有用于交换命令和响应的报文格式吗？

Q2-26　在文件传输期间，FTP 有用于交换文件或目录/文件列表的报文格式吗？

Q2-27　在 FTP 中能够只有控制连接而没有数据连接吗？请解释。

Q2-28　在 FTP 中能够只有数据连接而没有控制连接吗？请解释。

Q2-29　假设我们需要使用 FTP 下载一个音频。我们可以在命令中指定什么文件类型？

Q2-30　HTTP 和 FTP 都可以从服务器获取文件。当我们下载文件时使用什么协议？

Q2-31　HELO 和 MAIL FROM 命令在 SMTP 中都是必要的吗？为什么？

Q2-32　在图 2-20 的文字中，信封中的 MAIL FROM 与头部的 FROM 有什么区别？

Q2-33　Alice 在长途旅行中，没有查看她的电子邮件。之后她发现丢失了一些朋友们声称已经发给她的电子邮件或附件。这其中的问题可能是什么？

Q2-34　假设一个 TELNET 客户使用 ASCII 表示字符，但是 TELNET 服务器使用 EBCDIC 代表字符。当两者字符表示不同，客户如何登录到服务器上？

Q2-35　TELNET 应用程序中没有像 FTP 或 HTTP 中允许用户传输文件或访问页面的命令。这种应用程序在哪个方面有用？

Q2-36　一台主机能够使用一个 TELNET 客户获得 FTP 或 HTTP 等其他客户–服务器应用所提供的服务吗？

Q2-37　在 DNS 中，以下哪一项是 FQDN，哪一项是 PQDN？

a．xxx　　b．xxx.yyy.net　　c．zzz.yyy.xxx.edu.

Q2-38　在基于 DHT 的网络中，假设 $m = 4$。如果一个结点标识符的散列值为 18，这个结点在 DHT 空间中的位置在哪里？

Q2-39　在基于 DHT 的网络中，假设结点 4 有一个文件的关键字为 18。最接近关键字 18 的结点是 20。这个文件存储在哪里？

a．按照直接方法　　b．按照间接方法

Q2-40　在 Chord 网络中，我们有结点 N5 和关键字 k5。N5 是 k5 的前向结点还是后向结点？

Q2-41　在 Kademlia 网络中，标识符空间的大小为 1024。二叉树的高度（树根到叶结点的距离）是多少？每

个结点的子树数量是多少？每个路由表有多少行？

Q2-42　在 Kademlia 中，假设 *m* = 4 并且活动结点是 N4、N7 和 N12。k3 存储在系统的哪个位置？

思考题

P2-1　假设一个服务器的域名是 www.common.com。

a．给出一个想要获取文件 **/usr/users/doc** 的 HTTP 请求。客户接收 MIME 1 版、GIF 或 JPEG 图片，但是文档的存在时间不应该超过 4 天。

b．给出与 a 成功请求对应的 HTTP 响应。

P2-2　在 HTTP 中，画一幅图来给出 cookie 的应用，cookie 的应用场景是服务器只允许注册用户访问服务器。

P2-3　在 HTTP 中，画一幅图来给出网络门户中 cookie 的应用，请使用两个站点来表示。

P2-4　在 HTTP 中，画一幅图来给出 cookie 的应用，cookie 的应用场景是服务器使用 cookie 来做广告，请只用三个站点。

P2-5　请画图给出代理服务器的使用，代理服务器是客户网络的一部分：

a．给出当响应存储在代理服务器时，客户、代理服务器以及目标服务器之间的事务。

b．给出当响应不存储在代理服务器时，客户、代理服务器以及目标服务器之间的事务。

P2-6　在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有表示层。但是如果需要，应用层协议可以包含某些表示层定义的特征。HTTP 有表示层特征吗？

P2-7　HTTP 1.1 版将持续连接定义为默认连接。使用 RFC2616 找出客户或服务器如何将这个默认设置改为非持续连接？

P2-8　在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有会话层。但是如果需要，应用层协议可以包含某些会话层定义的特征。HTTP 有会话层特征吗？

P2-9　在 SMTP 中，一个发送者发送未格式化文本，请给出 MIME 头部。

P2-10　在 SMTP 中：

a．一个非 ASCII 码 1000 字节的报文使用 base64 编码。编码报文中有多少字节？有多少冗余字节？冗余字节与整个报文的比是多少？

b．一个非 ASCII 码 1000 字节的报文使用引用可打印编码。报文包含 90%ASCII 字符和 10%非 ASCII 字符。编码报文中有多少字节？有多少冗余字节？冗余字节与整个报文的比是多少？

c．比较前两个情况的结果。如果报文是 ASCII 和非 ASCII 字符报文的组合，效率提高多少？

P2-11　请用 base64 编码如下报文：

01010111 00001111 11110000

P2-12　请用引用可打印编码如下报文：

01001111 10101111 01110001

P2-13　根据 RFC1939，POP3 会话是以下四种状态之一：关闭、认证、事务或更新。画图给出这四种状态以及 POP3 如何在这些状态之间移动。

P2-14　POP3 协议有一些基本命令（每个客户−服务器需要实现）。使用 RFC1939 中的信息，找出以下基本命令的含义和用法：

a．STAT　　**b**．LIST　　**c**．DELE 4

P2-15　POP3 协议有一些可选命令（每个客户−服务器需要实现）。使用 RFC1939 中的信息，找出以下可选命令的含义和用法：

a．UIDL　　**b**．TOP 1 15　　**c**．USER　　**d**．PASS

P2-16　使用 RFC1939，假设 POP3 客户处于下载保持状态。如果服务器只从服务器下载两个 192 字节和 300 字节报文，请给出客户和服务器之间的事务。

P2-17　使用 RFC1939，假设 POP3 客户处于下载保持状态。如果服务器只从服务器下载两个 230 字节和 400 字节报文，请给出客户和服务器之间的事务。

P2-18　在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有表示层。但是如果需要，应用层协议可以包含某些表示层定义的特征。SMTP 有表示层特征吗？

P2-19　在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有会话层。但是如果需要，应用层协议可以包含某些会话层定义的特征。SMTP 或 POP3 有会话层特征吗？

P2-20 在 FTP 中，假设用户名为 John 的客户需要在服务器的目录 **/top/videos/general** 上存储一个名为 video2 的视频片段。如果选择临时端口号 56002，给出服务器和客户之间的交换的命令和响应。

P2-21 在 FTP 中，一个用户（Jane）想要使用临时端口 61017 从/usr/users/report 目录下获取一个名为 huge 的 EBCDIC 文件。文件很大，以至于用户想要在传输前压缩它。给出所有命令和响应。

P2-22 在 FTP 中，一个用户（Jan）想要在目录/usr/usrs/letters 下创建一个名为 Jan 的新目录。给出所有命令和响应。

P2-23 在 FTP 中，一个用户（Maria）想要将一个名为 file1 的文件从/usr/users/report 目录移动至/usr/top/letters 目录下。注意，这里有文件被重命名的情况。我们首先需要给出旧文件名然后定义新名称。给出所有命令和响应。

P2-24 在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有表示层。但是如果需要，应用层协议可以包含某些表示层定义的特征。FTP 有表示层特征吗？

P2-25 在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有会话层。但是如果需要，应用层协议可以包含某些会话层定义的特征。FTP 有会话层特征吗？

P2-26 在 Chord 中，假设标识符空间大小为 16。活动结点是 N3、N6、N8 以及 N12。给出 N6 的指针表（只给出目标关键字和后向结点列）。

P2-27 在 Chord 中，假设 N12 的后向结点是 N17。N12 是以下哪个关键字的前向结点。

a．k12　　**b**．k15　　**c**．k17　　**d**．k22

P2-28 在使用 DHT 的 Chord 网络中，$m = 4$，请画出标识符空间，其中放置 4 个对等结点，它们的 ID 是 N3、N8、N11 以及 N13，以及三个关键字 k5、k9 以及 k14。请判断哪个结点负责哪个关键字。为每个结点创建一个指针表。

P2-29 在 Chord 网络中，$m = 4$，结点 N2 指针表的值如下：N4、N7、N10 以及 N12。对于以下每个关键字，首先查找 N2 是否是关键的前向结点。如果不是，应该联系哪个结点（最接近前向结点）来帮助 N2 找到前向结点。

a．k1　　**b**．k6　　**c**．k9　　**d**．k13

P2-30 重复文中的例 2.16，但是假设结点 N12 需要找到负责关键字 k7 的结点。提示：请记住在模 32 运算中需要进行区间检测。

P2-31 重复文中的例 2.16，但是假设结点 N5 需要找到负责关键字 k16 的结点。提示：请记住在模 32 运算中需要进行区间检测。

P2-32 在 Pastry 中，假设地址空间是 16 并且 $b = 2$。地址空间中有多少位？列出一些标识符。

P2-33 在 Pastry 中 $m = 32$ 并且 $b = 4$，路由表以及叶子结点集的大小是多少？

P2-34 Pastry 的地址空间是 16 并且 $b = 2$，给出路由表的大纲。给出结点 N21 路由表中每个单元格的可能表项。

P2-35 在使用 DHT 的 Pastry 网络中，$m = 4$ 且 $b = 2$，请画出游 4 个结点 3 个关键字的标识符空间，4 个结点是 N02、N11、N20 以及 N23，3 个关键字是 k00、k12 以及 k24。请判断哪个结点负责哪个关键字。也请给出每个结点的叶子结点集和路由表。尽管它不是真实的，但是假设每两个结点间的接近度是基于数值接近程度度量的。

P2-36 在之前的问题中，请回答如下问题：

a．给出结点 N02 如何回应一个寻找负责 k24 结点的请求。

b．给出结点 N20 如何回应一个寻找负责 k12 结点的请求。

P2-37 使用文中图 2-55 的二叉树，给出结点 N11 的子树。

P2-38 使用文中图 2-56 中的路由表，如果结点 N0 接收到一个查询报文，它寻找负责 k12 的结点，请解释并给出路由。

P2-39 在 Kademlia 网络中 $m = 4$，我们有 5 个活动结点：N2、N3、N7、N10 以及 N12。找出每个活动结点的路由表（只有一列）。

P2-40 如文中图 2-60 所示，服务器如何知道一个客户已经请求了服务？

P2-41 如文中图 2-62 所示，服务器端如何为数据传输创建一个套接字？

P2-42 假设我们想要使表 2-25 中的 TCP 客户程序更具通用性，从而能够发送字符串并处理从服务器接收到的响应，如何做到这一点？

2.8　模拟实验

Applets

我们构建了一些 Java 小程序用于展示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些实际的协议。

实验作业

在第 1 章，我们下载并安装了 Wireshark，学习它的基本特性。在本章，我们使用 Wireshark 来捕获并研究一些应用层协议。这里使用 Wireshark 来模拟六种协议：HTTP、FTP、TELNET、SMTP、POP3 以及 DNS。

Lab2-1　在第 1 个实验中，我们使用 HTTP 获取网页。我们使用 Wireshark 来捕获分组进行分析。我们学习最普通的 HTTP 报文。我们也捕获响应报文并分析它们。在实验期间，我们也研究分析一些 HTTP 头部。

Lab2-2　在第 2 个实验中，我们使用 FTP 传输一些文件。我们使用 Wireshark 捕获分组。我们展示出 FTP 使用两个分离的连接：一个控制连接和一个数据传输连接。每次文件传输活动中数据连接都会被打开然后关闭。我们也展示出 FTP 是一个不安全的文件传输协议，因为每项事务都是用明文完成的。

Lab2-3　在第 3 个实验中，我们使用 Wireshark 捕获 TELNET 协议交换的分组。如 FTP 中一样，我们能够在被捕获的分组里观察到会话中的命令和响应。正如 FTP、TELNET 易受攻击，因为它用明文发送包括口令在内的所有数据。

Lab2-4　在第 4 个实验中，我们研究运行中的 SMTP 协议。我们发送一个电子邮件，使用 Wireshark 研究客户和服务器之间交换的 SMTP 分组的内容和格式。我们检验文中讨论的 SMTP 会话中的三个阶段。

Lab2-5　在第 5 个实验中，我们研究 POP3 协议的状态和行为。我们获取存储在 POP3 服务器邮箱中的邮件，通过分析那些经过 Wireshark 的报文来观察并分析 POP3 的状态以及交换报文的类型和内容。

Lab2-6　在第 6 个实验中，我们分析 DNS 协议的行为。除了 Wireshark，许多网络实用工具可用于查找 DNS 服务器中存储的信息。在这个实验中，我们使用 dig 命令（被 nslookup 所代替）。我们也使用 ipconfig 来管理主机中缓存的 DNS 记录。当我们使用这些工具时，我们用 Wireshark 来捕获这些工具发送的分组。

2.9　编程作业

利用你选择的编程语言，编写源代码，编译并测试如下程序：

Prg2-1　编写程序，使表 2-22 中的 UDP 服务器程序更通用：接收请求、处理请求并发回响应。

Prg2-2　编写程序，使表 2-23 中的 UDP 客户程序更通用：能够发送客户程序创建的任何请求。

Prg2-3　编写程序，使表 2-24 中的 TCP 服务器程序更通用：接收请求、处理请求并发回响应。

第 3 章

Computer Networks: A Top-Down Approach

传　输　层

TCP/IP 协议簇中的传输层位于应用层和网络层之间。它为应用层提供服务，并接收来自网络层的服务。传输层是客户程序和服务器程序之间的联络人，是一个进程到进程的连接。传输层是 TCP/IP 协议簇中的核心；它是因特网上从一点到另一个点传输数据的端到端逻辑传输媒介。这一章有些冗长，因为我们需要讲几个问题并介绍一些新概念。

- 3.1 节介绍通常从传输层获得的一般服务，如进程到进程通信、寻址、多路复用、多路分解、差错、流以及拥塞控制。
- 3.2 节讨论普通的传输层协议，如停止-等待协议、回退 *N* 帧协议以及选择重复协议。这些协议关注实际传输层协议所提供的传输流和差错控制服务。理解这些协议可以帮助我们更好地理解 UDP 和 TCP 协议的设计，以及其设计背后的逻辑。
- 3.3 节关注 UDP，这是本章讨论的两个协议中较简单的一个。UDP 缺乏许多我们想要从传输层协议得到的服务，但是正如我们所给出的，它的简单性对一些应用很有吸引力。
- 3.4 节讨论 TCP。我们首先列出它的服务和特性。然后我们使用转换图来给出 TCP 如何提供面向连接的服务。最终，我们使用 TCP 中的抽象窗口给出流和差错控制。之后讨论 TCP 中的拥塞控制，这个主题会在第 4 章网络层中重新讨论。我们也会给出一个列表以及 TCP 计时器的功能，它们在 TCP 提供的不同服务中都有所使用。本书网站列出的一些 TCP 常用的选项可用于深入研究。

3.1　介绍

传输层位于应用层和网络层之间。它在两个应用层之间提供进程到进程服务，一个进程在本地主机，另一个在远程主机。使用逻辑连接提供通信，意味着两个应用层可以位于地球上的不同位置，两个应用层假设存在一条想象的直接连接，通过这条连接它们可以发送和接收报文。图 3-1 给出了逻辑连接背后的思想。

这幅图给出的场景和我们在第 2 章里表示应用层的场景（图 2-1）相同。Alice 那台位于 Sky Research 公司的主机在传输层创建了一条逻辑连接，它连接到 Bob 在 Scientific Books 公司的主机。两个公司在传输层通信，好像它们之间有一条真正的连接。图 3-1 展示出只有两个终端系统（Alice 和 Bob 的计算机）使用传输层服务；所有中间路由器只使用前三层。

传输层服务

正如我们在第 1 章讨论的，传输层位于网络层和应用层之间。传输层负责向应用层提供服务；它接收来自网络层的服务。在这一节，我们讨论传输层提供的服务；在下一节，我们讨论几个传输层协议。

进程到进程通信

传输层协议的首要任务是提供**进程到进程通信**（process-to-process communication）。进程是使用传输层服务的应用层实体（运行着的程序）。我们在讨论进程到进程通信如何实现之前，需要理解主机到主机通信与进程到进程通信的不同之处。

HZ BOOKS 华章图书

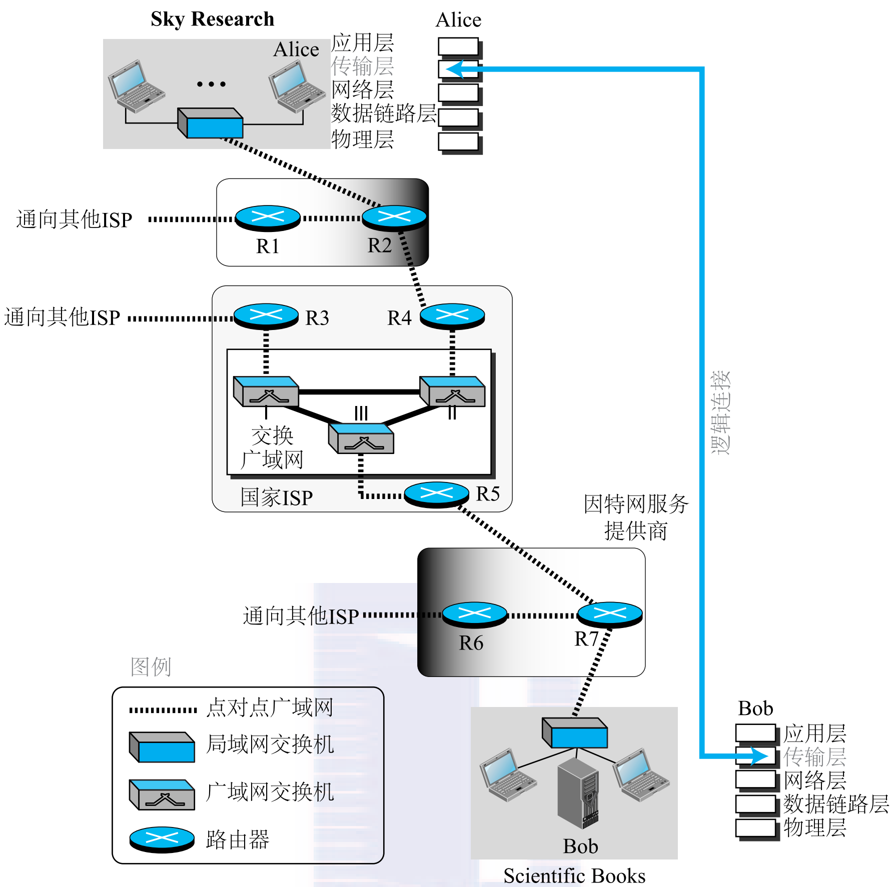


图 3-1　传输层的逻辑连接

网络层（第 4 章讨论）负责计算机层次的通信（主机到主机通信）。网络层协议只把报文传递到目的计算机。然而，这是不完整的传递。报文仍然需要递交给正确的进程。这正是传输层接管的部分。传输层协议负责将报文传输到正确的进程。图 3-2 给出了网络层和传输层的范围。

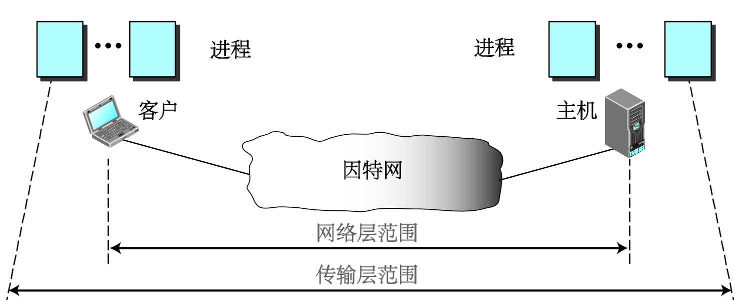


图 3-2　网络层与传输层

寻址：端口号

尽管有一些方法可以实现进程到进程通信，但是最常用的是通过**客户–服务器模式**（client-server paradigm，见第 2 章）。本地主机上的进程称为客户，它通常需要来自远程主机上的进程提供的服务，这种远程主机称为服务器。

这两个进程（客户和服务器）有相同的名字。例如，如果要从远程机器上获得日期和时间，我们需要在本地主机上运行 Daytime 客户进程并在远程机器上运行 Daytime 服务进程。

然而，目前的操作系统支持多用户和多程序运行的环境。一个远程计算机在同一时间可以运行多个服务器程序，就像许多本地计算机可在同一时间运行一个或多个客户应用程序一样。对通信来说，我们必须定义本地主机、本地进程、远程主机以及远程进程。我们使用 IP 地址来定义本地主机和远程主机（将在第 4 章讨论）。为了定义进程，我们需要第二个标识符，称为**端口号**（port number）。在 TCP/IP 协议簇中，端口号是在 0 到 65 535 之间的 16 位整数。

客户程序用端口号定义它自己，这称为**临时端口号**（ephemeral port number）。临时这个词表示短期的（short-lived），它之所以被使用是因为客户的生命周期通常很短。为了客户–服务器程序能正常工作，临时端口号推荐值为大于 1 023。

服务器进程必须使用一个端口号定义它自己。但是，这个端口号不能随机选择。如果服务器站点的计算机运行一个服务器进程，并随机分配一个数字作为端口号，那么在客户站点的进程想访问该服务器并使用其服务时，将不知道此端口号。当然，一个解决方法是发送一个特殊分组，并请求一个特定服务器的端口号，但是这需要更多的开销。TCP/IP 决定使用全局端口号；它们称为**熟知端口号**（well-known port number）。但是这条规则也有例外，例如，有一些客户端也被分配了熟知端口号。每一个客户进程都知道相应服务器进程的熟知端口号。例如，前面所讨论的 Daytime 客户进程可以使用临时（暂时）端口号 52 000 来表示自己，但服务器 Daytime 进程必须使用熟知（永久）端口号 13。图 3-3 表示了这一概念。

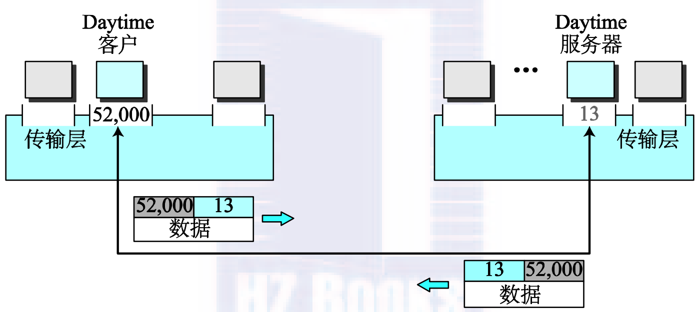


图 3-3　端口号

现在应该搞清楚，在选择数据的最终目的端时，IP 地址和端口号起着不同的作用。目的 IP 地址在世界范围的不同主机中确定一个主机。但主机被选定后，端口号定义了在该特定主机上的多个进程中的一个进程（见图 3-4）。

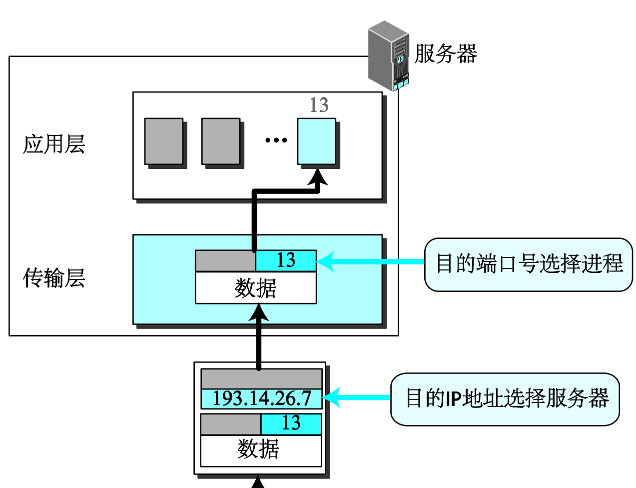


图 3-4　IP 地址和端口号

ICANN 范围

ICANN（见附录 D）已经把端口号编码划分为三种范围：熟知的、注册的和动态的（或私有的），如图 3-5 所示。

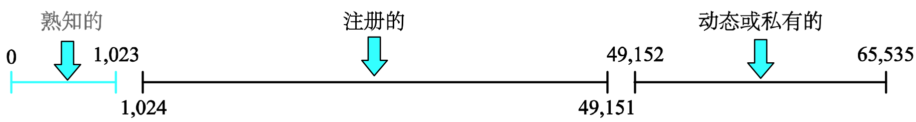


图 3-5　ICANN 范围

- **熟知端口**。端口号的范围是 0~1023，由 ICANN 分配和控制。这些是熟知端口号。
- **注册端口**。端口号的范围是 1024~49151，ICANN 不分配也不控制。它们可在 ICANN 注册以防重复。
- **动态端口**。端口号的范围是 49152~65535。这一范围内的端口号既不受控制又不需要注册，可以由任何进程使用。它们是临时或私有端口号。

例 3.1　在 UNIX 中，熟知端口号存储在/etc/services 文件中，这个文件的每行给出服务器名和熟知端口号。我们可用 grep 命令提取该行所对应的应用。下面表示了 TFTP 的端口。注意，TFTP 可使用 UDP 或 TCP 的端口 69。

```
$grep        tftp/etc/services
tftp 69/tcp
tftp 69/udp
```

SNMP（见第 9 章）使用两个端口号（161 和 162），每一个用于不同目的。

```
$grep        snmp/etc/services
snmp161/tcp#Simple Net Mgmt Proto
snmp161/udp#Simple Net Mgmt Proto
snmptrap162/udp#Traps for SNMP
```

套接字地址

在 TCP 协议簇中的传输层协议需要 IP 地址和端口号，它们各在一端建立一条连接。一个 IP 地址和一个端口号结合起来称为**套接字地址**（socket address）。客户套接字地址唯一定义了客户进程，而服务器套接字地址唯一地定义了服务器进程（见图 3-6）。

为了使用因特网中的传输层服务，我们需要一对套接字地址：客户套接字地址和服务器套接字地址。这四条信息是网络层分组头部和传输层分组头部的组成部分。第一个头部包含 IP 地址，而第二个头部包含端口号。

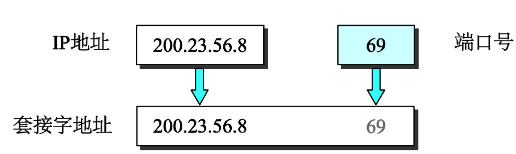


图 3-6　套接字地址

封装与解封装

为了将报文从一个进程发送到另一进程，传输层协议负责封装与解封装报文（见图 3-7）。封装在发送端发生。当进程有报文要发送，它将报文与一组套接字地址和其他信息一起发送到传输层，这依赖于传输层协议。传输层接收数据并加入传输层头部。因特网中传输层的分组称为用户数据报（user datagram）、段（segment）或分组（packet），这取决于我们使用什么传输层协议。在一般讨论中，我们将传输层有效载荷称为分组。

解封装发生在接收端。当报文到达目的传输层，头部被丢弃，传输层将报文传递到应用层运行的进程。如果需要响应接收到的报文，发送方的套接字地址被发送到进程。

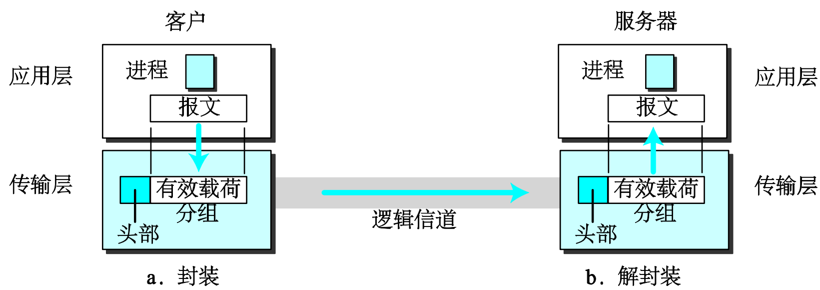


图 3-7 封装与解封装

多路复用与多路分解

每当一个实体从一个以上的源接收到数据项时，称为**多路复用**（multiplexing，多对一）；每当一个实体将数据项传递到一个以上的源时，称为**多路分解**（demultiplexing，一对多）。源端的传输层执行复用；目的端的传输层执行多路分解（见图 3-8）。

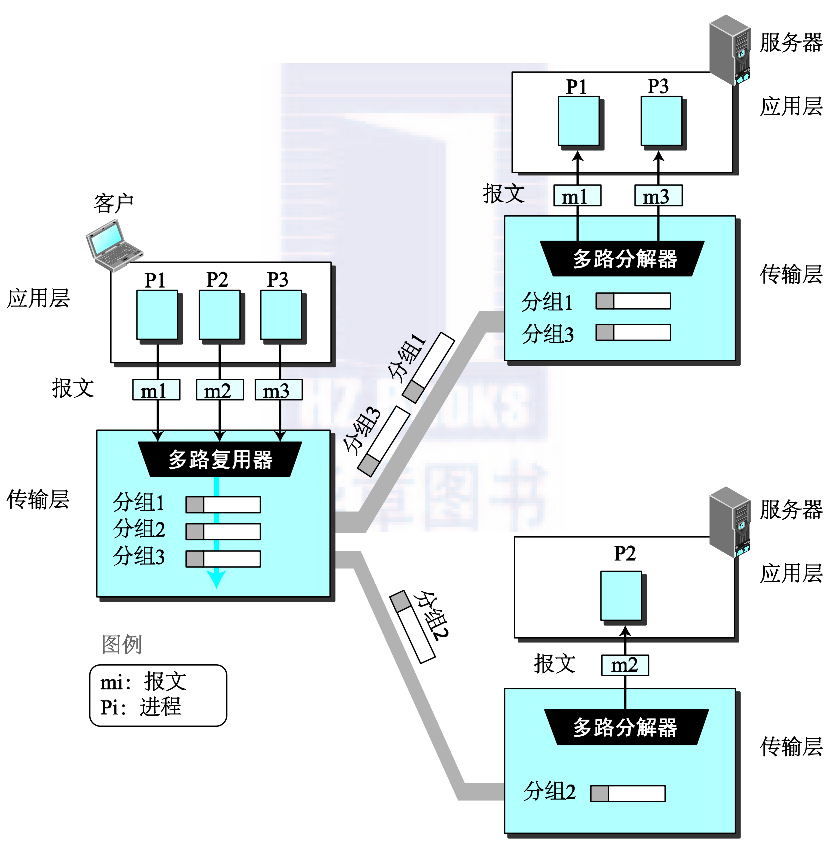


图 3-8 多路复用和多路分解

图 3-8 给出了一个客户和两个服务器之间的通信。客户端运行三个客户进程：P1、P2 和 P3。进程 P1 和 P3 需要将请求发送到对应的服务器进程。客户进程 P2 需要将请求发送到位于另外一个服务器的服务器进程。客户端的传输层接收到来自三个进程的三个报文并创建三个分组。它起到了多路复用器的作用。分组 1 和 3 使用相同的逻辑信道到达第一个服务器的传输层。当它们到达服务器时，传输层起到多路分解器的作用并将报文分发到两个不同的进程。第二个服务器的传输层接收分组 2 并将它传递到相应的进程。注意，尽管只有一个报文，我们仍然用到多路分解。

流量控制

每当一个实体创建数据项并且有另一个实体消耗它们时，就存在生产速率和消费速率的平衡问题。如果数据项生产比消费快，那么消费者可能被淹没并且可能要丢弃一些数据项。如果数据项生产比消费慢，那么消费者必须等待，系统就会变得低效。流量控制与第一种情况相关。我们需要在消费者端防止丢失数据项。

推或拉

从生产者传递数据项到消费者有两种方式：推（push）或拉（pull）。每当发送方生产数据项时，它无须事前获得消费者的请求就会发送它们——这种传递称为推。如果生产者在消费者请求这些数据项之后进行发送，这种传递称为拉。图 3-9 给出了这两种传递类型。

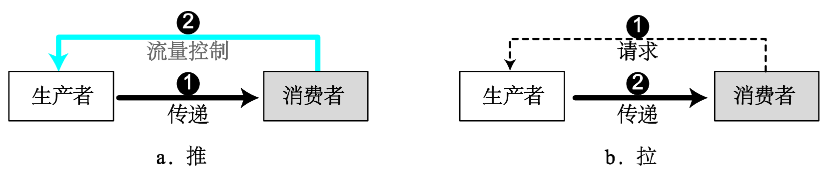


图 3-9　推或拉

当生产者推数据项时，消费者可能被淹没并需要相反方向的流量控制，以此来防止丢弃这些数据项。换言之，消费者需要警告生产者停止传递，并且当消费者再次准备好接收数据时通知生产者。当消费者拉数据项，它会在自身做好准备时进行请求。在这种情况下，不需要流量控制。

传输层流量控制

在传输层通信中，我们需要处理四个实体：发送方进程、发送方传输层、接收方传输层和接收方进程。应用层的发送方进程仅仅是一个生产者。它生产报文块，并把它们推到传输层。发送方传输层有两个作用：它既是消费者也是生产者。它消费生产者推来的报文。它将报文封装进分组并传递到接收方传输层。接收方传输层也有两个作用：它是消费者，消费从发送方那里接收来的分组；它也是生产者，解封装报文并传递到应用层。然而，最后的传递通常是拉传递；传输层等待直到应用层进程请求报文。

图 3-10 展示出，我们至少需要两种流量控制：从发送方传输层到发送方应用层的流量控制和从接收方传输层到发送方传输层的流量控制。

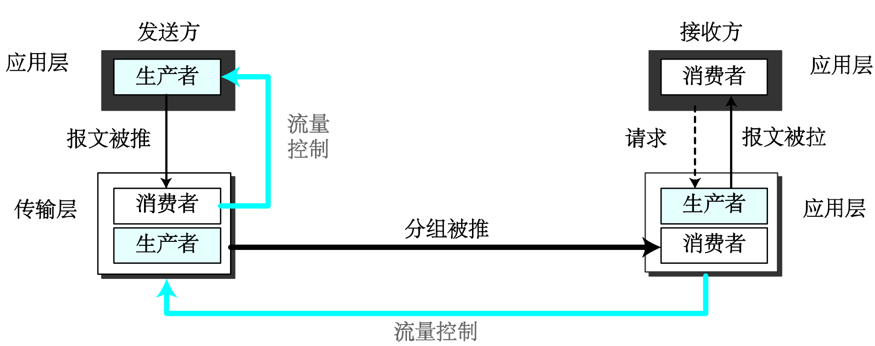


图 3-10　传输层流量控制

缓冲区

尽管流量控制可以用多种方式实现，但通常的方式是使用两个缓冲区：一个位于发送方传输层，另一个位于接收方传输层。缓冲区是一组内存单元，它可以在发送端和接收端存储分组。消费者向生产者发送信号从而进行流量控制通信。

当发送方传输层的缓冲区已满，它就通知应用层停止传输报文块；当有空闲位置时，它通知应用层可以再次传输报文块。

当接收方传输层的缓冲区已满，它就通知发送方传输层停止传输分组；当有空闲位置时，它通知发送方传输层可以再次传输分组。

例 3.2　上述讨论要求消费者与生产者在以下两种情况下进行通信：缓冲区满或缓冲区有空闲位置。如果双方使用的缓冲区只有一个存储单元，那么通信就会变得更简单。假设每个传输层使用一个存储单元来存储分组。当位于传输层的这个存储单元为空时，发送方传输层就会向应用层发送一个通知，要求发送下一个块；当位于接收方传输层的这个存储单元为空时，它向发送方传输层发送一个确认，要求发送下一个分组。然而，我们后文将会看到，这种流量控制在发送方和接收方只使用含有一个存储单元的缓冲区，非常低效。

差错控制

在因特网中，由于网络层（IP）是不可靠的，如果应用层需要可靠性，我们需要使传输层变得可靠。可靠性可以通过在传输层加入差错控制服务来实现。传输层的差错控制负责以下几个方面：

1. 发现并丢弃被破坏的分组。
2. 记录丢失和丢弃的分组并重传它们。
3. 识别重复分组并丢弃它们。
4. 缓冲失序分组直到丢失的分组到达。

差错控制不像流量控制，它仅涉及发送方和接收方传输层。我们假设在应用层和传输层之间交换的报文块是不会产生差错的。图 3-11 给出了发送方和接收方传输层的差错控制。正如传输控制，大多数情况下，接收方传输层管理差错控制，它通过告知发送方传输层存在问题来进行管理。

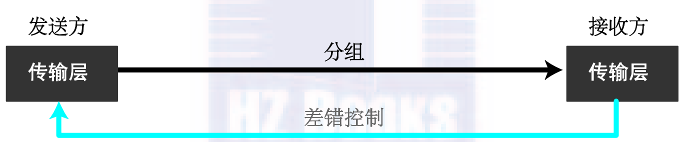


图 3-11　传输层的差错控制

序号

差错控制需要发送方传输层知道哪个分组要被重传并且接收方传输层需要知道哪个分组是重复的、哪个分组是失序的。如果分组是编号的，这个就可以实现。我们可以在传输层分组中加入一个字段来保存分组的**序号**（sequence number）。当分组被破坏或丢失，接收方传输层可按某种方式通知发送方传输层去利用序号重传分组。如果两个接收到的分组具有相同的序号，接收方传输层也能发现重复分组。可以通过观察序号的间隔辨别失序分组。

分组一般按序编号。然而，由于我们需要在头部包含每个分组的序号，因此需要设置一个界限。如果分组的头部允许序号最多为 m 比特位，那么序号范围就是 0 到 2^m-1。例如，如果 m 是 4，序号范围是 0 到 15 的闭区间。然而，我们可以回绕。因此，这种情况下，序号为

0，1，2，3，4，5，6，7，8，9，10，11，12，13，14，15，0，1，2，3，4，5，6，7，8，9，10，11，…

换言之，序号是模 2^m 的。

> 对于差错控制，序号是模 2^m 的，这里 m 是序号字段的大小，单位是比特。

确认

我们可以发送积极或消极的信号作为差错控制，但是我们只讨论积极信号，这在传输层中最常见。接收方可以为每一组正确到达的分组发送一个确认（ACK）。接收方可以简单地丢弃被破坏的分组。

发送方如果使用计时器，它就可以发现丢失分组。当一个分组被发送，发送方就开启一个计时器。如果 ACK 在计时器超时之前没有到达，那么发送方重发这个分组。重复的分组可以被接收方默默丢弃。失序的分组既可以被丢弃（被发送方当做丢失报文对待），也可以存储直到丢失的那个分组到来。

流量和差错控制的组合

我们已经讨论过，流量控制要求使用两个缓冲区，一个在发送端另一个在接收端。我们也已经讨论过差错控制要求两端均使用序号和确认号。如果我们使用两个带序号的缓冲区：一个位于发送端，一个位于接收端，那么这两个需要可以结合起来。

在发送端，当分组准备发送时，我们使用下一个缓冲区空闲位置号码 x 作为分组的序号。当分组被发送，一个分组的备份存储在内存位置 x，等待来自另一端的确认。当与被发送分组相关的确认到达时，分组被清除，内存位置空闲出来。

在接收端，当带有序号 y 的分组到达时，它被存储在内存位置 y 上，直到应用层准备好接收它。这时发送一个确认表明分组 y 的到达。

滑动窗口

由于序号进行模 2^m 操作，因此一个环可以代表从 0 到 2^m-1 的序号（见图 3-12）。缓冲区由一组片段代表，称为**滑动窗口**（sliding window），它随时占据环的一部分。在发送端，当一个报文被发送，相应的片段就被标记。当有所片段都被标记时，意味着缓冲区满且不能从应用层进一步接收报文。当确认到达时，相应片段被取消标记。如果从窗口开始处有一些连续的片段没有被标记，那么窗口滑过这些相应序号的范围，允许更多的片段进入窗口尾部。图 3-12 给出发送方的滑动窗口。序号以 16 为模（$m=4$）且窗口大小为 7。请注意滑动窗口仅仅是一个抽象：实际情况是使用计算机变量来保存下一个和最后一个待发送的分组。

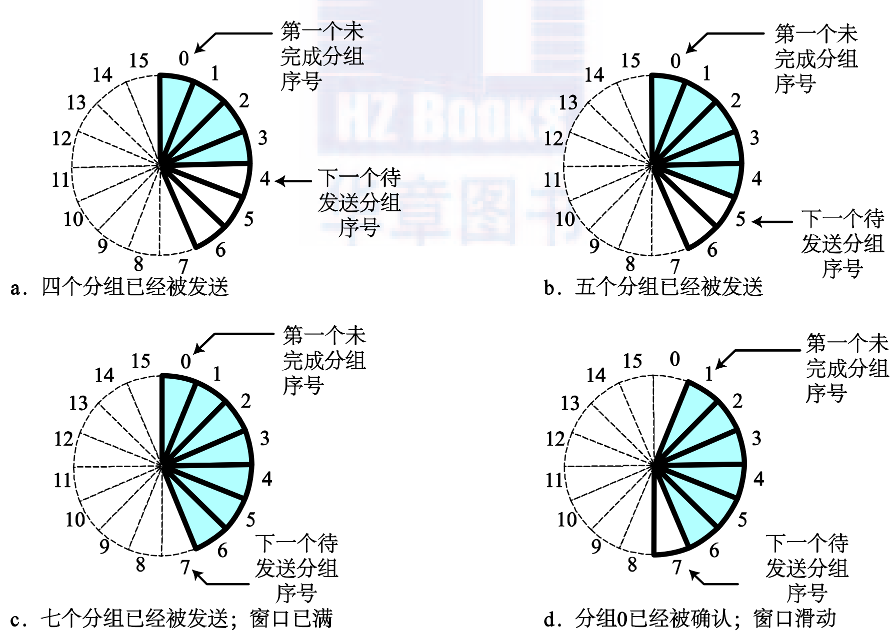


图 3-12　圆形滑动窗口

大多数协议使用线性形式来表示滑动窗口。虽然想法是相同的，但是它通常占用更少的页面空间。图 3-13 给出这种表示方法。这两种表示方法告诉我们相同的事情。如果拿起图 3-13 中每一幅图的两个端点，并且弯曲它们，我们就可以得到与图 3-12 中相同的图。

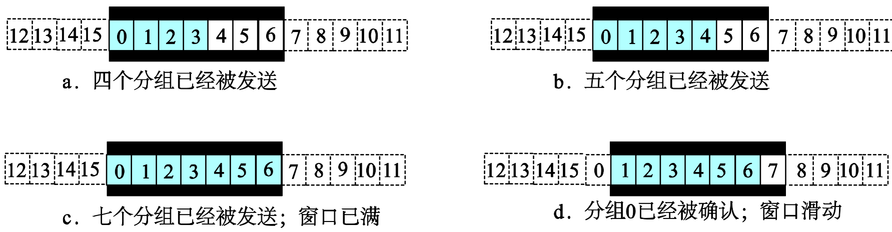


图 3-13　线性形式滑动窗口

拥塞控制

在因特网之类的分组交换网络中存在一个重要问题，这就是**拥塞**（congestion）。如果网络中的负载（load，即发送到网络的分组数）大于网络的容量（网络可以处理的分组数），那么网络就可能发生拥塞。**拥塞控制**（congestion control）指的是一种机制和技术，它控制拥塞并将负载保持在容量以内。

我们可能会问，为什么网络中会有拥塞。在任何系统中发生的拥塞都需要等待。例如，由于车流量不正常，如高峰时段的事故，高速公路上会发生拥塞导致交通堵塞。

在处理前后都存储分组的缓冲区时，由于路由器和交换机中有队列，网络或互联网会发生拥塞。比如，一个路由器，它的每一个接口都有一个输入队列和一个输出队列。如果路由器不能以分组到达的速率进行处理，队列就会超载，拥塞就会发生。传输层的拥塞实际上是网络层拥塞的结果。我们在第 4 章讨论网络层拥塞及其成因。本章后面，假设网络层没有拥塞控制，我们给出 TCP 如何实现它自己的拥塞控制机制。

无连接和面向连接服务

传输层协议就像网络层协议一样，可以提供两种类型的服务：无连接服务和面向连接服务。然而，这些传输层服务的本质与网络层不同。在网络层，无连接服务可能意味着属于同一个报文的不同数据报有不同路径。在传输层，我们不关心分组的物理路径（我们假设两个传输层之间有一条逻辑连接）。传输层的无连接服务意味着分组之间的独立；面向连接服务意味着依赖。接下来让我们仔细研究这两种服务。

无连接服务

在无连接服务中，源进程（应用程序）需要将报文分成传输层可接受大小的数据块，并把它们一个一个地传递到传输层。传输层将每一个数据块看做彼此没有关系的单元。当一个块从应用层到达时，传输层将其封装在分组中并发送。为了展示分组的独立，我们假设客户进程有三个报文块要发送给服务器进程。这些块被按序交给无连接传输协议。然而，由于传输层的这些分组之间没有联系，分组可能失序到达目的地并且被失序传递给服务器进程（见图 3-14）。

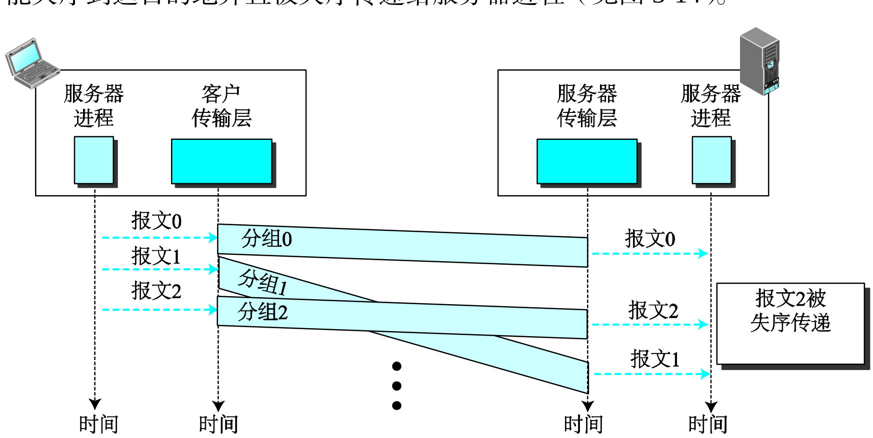


图 3-14　无连接服务

在图 3-14 中，我们使用时间轴给出分组的移动，但是我们假设进程到传输层的传递过程是瞬时的，传输层到进程的传递亦然。如图 3-14 所示，在客户端，三个报文块按序传递给客户传输层（0、1 和 2）。由于第二个分组在传输中的额外延迟，服务器报文的传递失序（0、2 和 1）。如果这三个数据块属于同一个报文，那么服务器进程可能会收到一个奇怪的报文。

如果一个分组丢了情况就更糟糕了。由于分组没有序号，接收方传输层不知道一个报文已经丢失。它仅仅将两个数据块传送到服务器进程。

以上两个问题是由于双方传输层没有互相协调所致。接收方传输层不知道第一个分组将要到来，也不知道所有的分组已经到来。

我们可以说，流量控制、差错控制以及拥塞控制都不能在无连接服务中有效实现。

面向连接服务

在面向连接的服务中，首先需要建立客户和服务器之间的逻辑连接。只有连接建立之后才能进行数据交换。在数据交换之后，连接需要拆除（见图 3-15）。

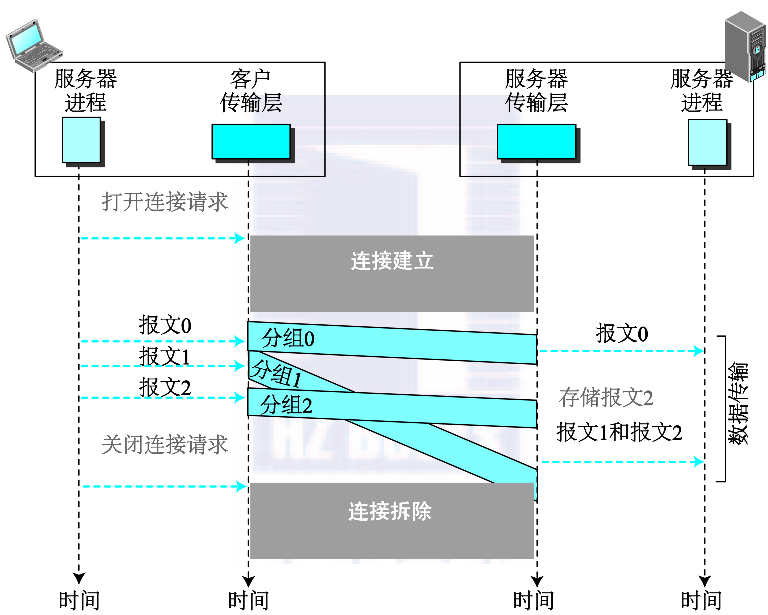


图 3-15　面向连接服务

如前所述，传输层的面向连接服务与网络层的面向连接服务不同。在网络层，面向连接服务意味着两个终端主机以及这之间的所有路由器都进行协调。在传输层，面向连接服务仅仅涉及两个主机；服务是端到端的。这意味着我们能够在传输层建立一个面向连接协议，其下的网络层可以是无连接协议也可以是面向连接协议。图 3-15 给出传输层面向连接服务中的连接建立、数据传输以及拆除阶段。

在面向连接协议中，我们可以实现流量控制、差错控制以及拥塞控制。

有限状态机

当提供无连接或面向连接服务时，传输层协议的行为都可以用**有限状态机**（finite state machine，FSM）来更好地表示。图 3-16 利用有限状态机给出了一幅传输层的图。使用这个工具，每个传输层（发送方或接收方）都被当做具有有限个状态的状态机来教授。这个状态机总是处于其中一种状态，直到一个事件（event）发生才改变状态。每个事件与两种反应相关：定义将要执行的动作列表（可能是空的），以及决定下一个状态（可能与当前状态相同）。必须定义一种状态为初始状态，这个状态是状态机开启时的状态。在图 3-16 中，我们使用圆角矩形来表示状态，用带灰度的文本来表示事件，用普通黑色文本来表示动作。尽管我们之后用斜线代替了水平线，但是此处使用水平

线来分割事件和动作。箭头表示移动到下一个状态。

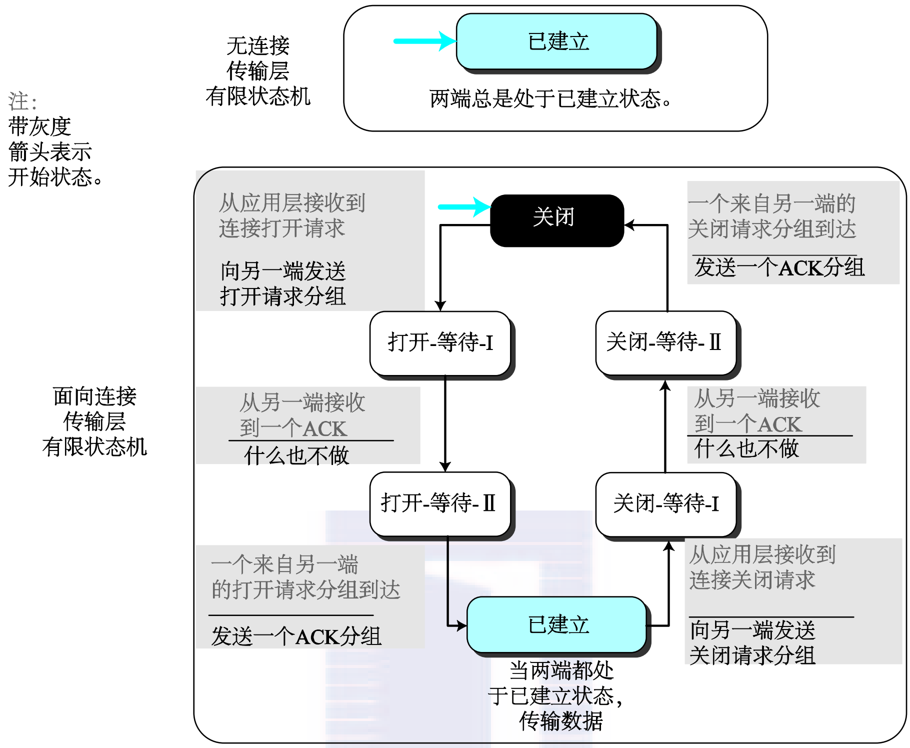


图 3-16　用 FSM 表示的无连接和面向连接服务

我们可以将无连接传输层看做只有一个状态的有限状态机：已建立状态。每一端的状态机（客户和服务器）总是处于已建立状态，准备发送并接收传输层分组。

换言之，在面向连接传输层的有限状态机中，在到达已建立状态之前需要经过三个状态。关闭连接之前状态机也需要经过三个状态。当不存在连接时，状态机处于关闭状态。它保持这个状态直到一个来自本地进程的打开连接的请求到达；状态机向远程传输层发送一个打开请求分组并将状态转移到打开-等待-I。当从另一端接收到确认，本地有限状态机进入打开-等待-Ⅱ状态。当状态机处于这个状态时，单向连接已经建立，但是如果需要建立双向连接，状态机需要在这个状态继续等待，直到另一端也请求连接。当请求被接收时，状态机发送一个确认并进入已建立状态。

当两个终端都处于已建立状态时，数据和数据确认可以在它们之间交换。然而，我们需要记住，无连接和面向连接传输层中的已建立状态都代表一组数据传输状态，我们将在下一节传输层协议中讨论这个问题。

为了拆除连接，应用层向本地传输层发送一个关闭请求分组。传输层向另一端发送一个关闭请求分组，并进入关闭-等待-I状态。当接收到来自另一端的确认时，状态机进入关闭-等待-Ⅱ状态，并且等待来自另一端的关闭请求分组。当这个分组到达时，状态机发送一个确认并进入关闭状态。

我们之后会讨论到，面向连接的有限状态机有很多种变化。我们也将看到有限状态机如何被压缩或扩展，以及状态名称如何变化。

3.2　传输层协议

我们通过将上一节描述的一组服务组合起来创建了传输层协议。为了更好地理解这些协议的行为，我们从一个最简单的协议开始并逐步深入。TCP/IP 协议或使用修改的传输层协议，或将几个

协议结合起来使用。我们在这一节讨论这些常见协议，为理解本章剩余的复杂协议铺平道路。为了使讨论简单，我们首先将这些协议作为单向协议（即单工）讨论，在单向协议中，数据分组沿着一个方向移动。在本章结尾部分，我们简要讨论它们如何变成双向协议，在双向协议中数据可以沿两个方向移动（即双工）。

3.2.1　简单协议

我们的第一个协议是一个简单的无连接协议，它既没有流量控制也没有差错控制。我们假设接收方能够立即处理它所收到的任何分组。换言之，接收方永远不会被接收到的分组淹没。图 3-17 给出了这种协议的框架。

发送方的传输层从发送方的应用层接收到报文，从中建立一个分组并发送它。接收方的传输层从网络层接收到这个分组，从分组中提取报文并传递到应用层。发送方和接收方的传输层都为应用层提供传输服务。



图 3-17　简单协议

有限状态机

直到应用层有报文待发送，发送端才发送分组。直到一个分组到达，接收端才将报文传递到它的应用层。我们可以使用两个有限状态机来表示这些要求。每个有限状态机只有一种状态，即准备状态（ready state）。发送方状态机保持准备状态，直到一个来自应用层进程的请求到来。当这个事件发生时，发送方状态机将报文封装在分组内，并将其发送到接收方状态机。接收方状态机保持准备状态，直到一个来自发送方状态机的分组到来。当这个事件发生时，接收方状态机从分组内解封装出报文，并将其发送到应用层进程。图 3-18 给出了简单协议的有限状态机。我们之后会看到，UDP 协议是这个协议的轻微改动版本。

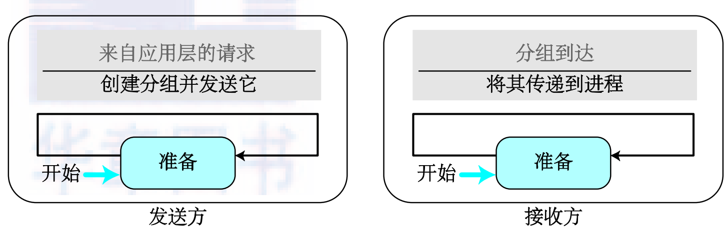


图 3-18　简单协议的有限状态机

例 3.3　图 3-19 给出了使用这种协议的通信示例。它非常简单。发送方一个接一个地发送分组，甚至不用考虑接收方能否承受。

3.2.2　停止–等待协议

我们的第二个面向连接协议称为**停止–等待协议**（Stop-and-Wait-protocol），它使用流量和差错控制。发送方和接收方都使用大小为 1 的滑动窗口。发送方在某一时刻发送一个分组，并且在发送下一个分组之前等待确认。为了发现被破坏分组，我们需要在每个数据分组中加入校验和。当一个分组到达接收端时，它就被检测。如果校验和不正确，分组就是被破坏的并被悄悄地丢弃。接收方的沉默对发送方来说是一种信号，即那个分组不是被破坏就是丢失了。每当发送方发送一个分组时，它都开启一个计时器。如果在计时器超时之前接收到确认，那么计时器就被关闭并且发送下一个分组（如果它有待发送分组）。如果计时器超时，发送方就认为分组丢失或被破坏，于是重发之前的分组。这意味着在确认到来之前，发送方都需要存储分组的副本。图 3-20 给出了停止-等待协议的框架。注意，信道中每次只能有一个分组或一个确认。

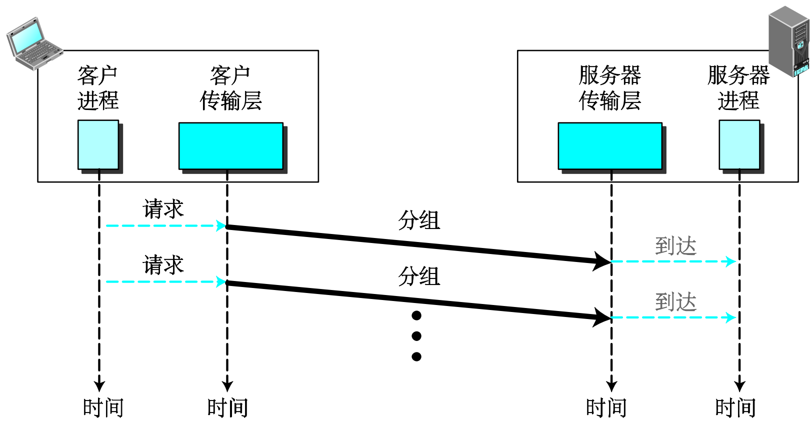


图 3-19　例 3.3 的流程图

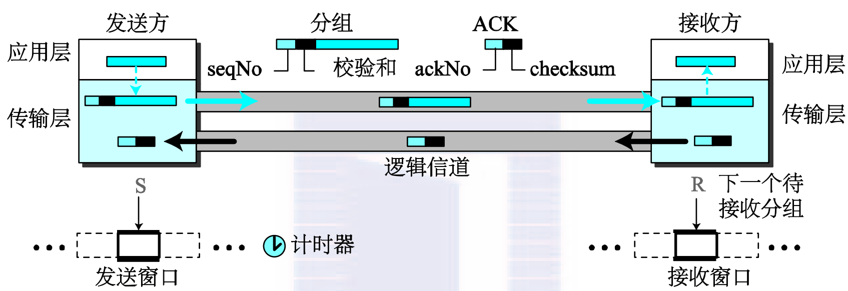


图 3-20　停止–等待协议

停止–等待协议是一个提供流量和差错控制的面向连接协议。

序号

协议使用序号和确认号来防止重复分组。一个字段被加入分组头部来保存那个分组的序号。一件需要着重考虑的事情就是序号的范围。由于想使分组大小最小化，所以我们寻找能提供无歧义通信的最小的序号范围。让我们来讨论一下所需要的序号范围。假设我们使用 x 作为序号；我们只需要在之后使用 $x+1$，不需要 $x+2$。为了表示这种情况，假设发送端已经发送了带有序号 x 的分组。可能发生三件事：

1. 分组安全完整地到达接收端；接收方发送一个确认。确认到达发送端，使发送端发送下一个序号为 $x+1$ 的分组。

2. 分组被破坏或未到达接收端；发送方在超时后重新发送分组（序号 x）。接收方返回一个确认。

3. 分组安全完整到达接收端；接收方发送一个确认，但是确认被破坏或丢失了。发送方在超时后重传分组（序号 x）。注意，这里分组是重复的。接收方可以认出这个事实，因为它等待分组 $x+1$，但是收到了分组 x。

我们可以看到，由于接收方需要区分情况 1 和 3，因此需要序号 x 和 $x+1$。但是不需要一个编号为 $x+2$ 的分组。在情况 1 中，分组可以再次被编号 x，由于分组 x 和 $x+1$ 被确认，两端都不会产生歧义。在情况 2 和 3 中，新的分组是 $x+1$ 而不是 $x+2$。如果仅仅需要 x 和 $x+1$，我们可以令 $x=0$ 且 $x+1=1$。这意味着序号是 0、1、0、1、0，等等。这称为模 2 运算。

确认号

由于序号必须适合于数据分组和确认，因此我们使用这种惯例：确认号总声明接收方预期接收的下一个分组（next packet expected）序号。例如，如果 0 号分组已经安全完整到达，接收方发送一个确认号为 1 的 ACK（意味着 1 号分组是预期接收的下一个分组）。如果 1 号分组已经安全完整

到达，接收方发送一个确认号为 0 的 ACK（意味着 0 号分组是预期接收的下一个分组）。

在停止–等待协议中，确认号总是以模 2 运算的方式声明预期接收的下一个分组序号。

发送方有一个控制变量，我们称之为 **S**（sender），它指向发送窗口中唯一的一个槽。接收方有一个控制变量，我们称之为 **R**（receiver），它指向接收窗口中唯一的一个槽。

有限状态机

图 3-21 给出了停止–等待协议的有限状态机。由于这个协议是面向连接的，因此在交换数据分组之前，两端都处于已建立状态。事实上，这些状态嵌套在已建立状态之中。

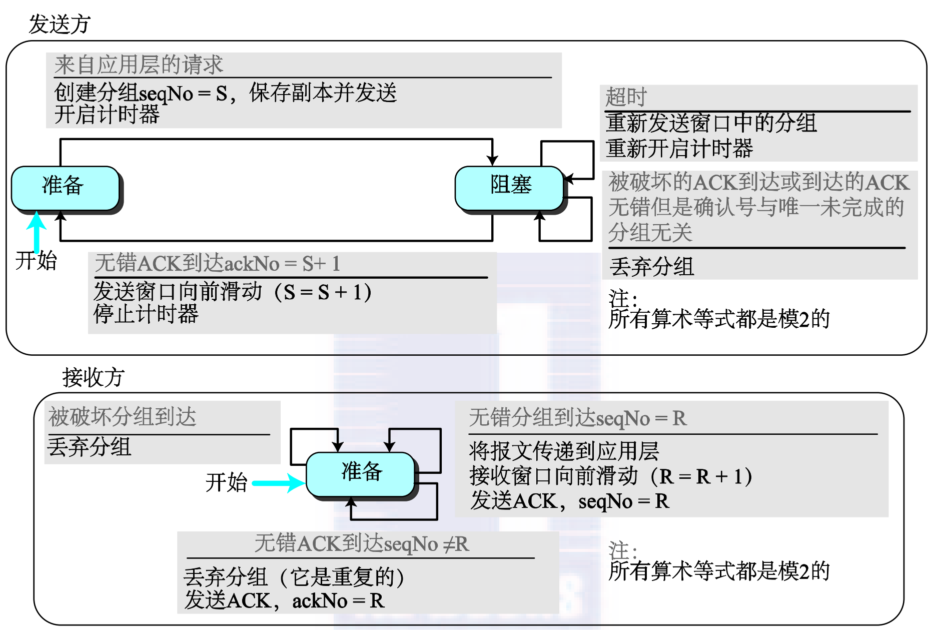


图 3-21 停止–等待协议有限状态机

发送方

初始时，发送方处于准备状态，当时它可以在准备状态和阻塞状态之间转换，变量 S 初始化为 0。

- **准备状态**。当发送方处于这种状态时，它只等待一个事件发生。如果来自应用层的请求到来，发送方创建一个分组，并将其序号设为 S。保存分组的副本，发送分组。之后，发送方开启唯一的计时器。发送方进入阻塞状态。
- **阻塞状态**。当发送方处于这个状态时，可能发生三个事件：

a．如果无错 ACK 到达，它的确认号与下一个待发送分组相关，这意味着 ackNo = (S + 1)modulo 2，然后关闭计时器。窗口滑动 S = (S + 1) modulo 2。最终发送方进入准备状态。

b．如果到达的是被破坏 ACK 或是 ackNo≠(S + 1) modulo 2 的无错 ACK，那么 ACK 被丢弃。

c．如果发生超时，发送方重新发送唯一的未完成分组并重新开启计时器。

接收方

接收方总是处于准备（ready）状态。可能发生三个事件：

a．如果 seqNo = R 的无错分组到达，分组中的报文被传递到应用层。之后窗口滑动，R = (R + 1) modulo 2。最终，确认号为 ackNo = R 的 ACK 被发送。

b．如果 seqNo≠R 的无错分组到达，分组被丢弃，但是确认号为 ackNo = R 的 ACK 被发送。

c．如果一个被破坏的分组到达，分组被丢弃。

例 3.4　图 3-22 给出了停止–等待协议的例子。分组 0 被发送且被确认。分组 1 丢失并在超时后重发。重发分组 1 被确认并且计时器停止。分组 0 被发送且被确认，但是确认丢失了。发送方不知道是分组丢失了还是确认丢失了，因此在超时之后，它重发分组 0，这次被确认了。

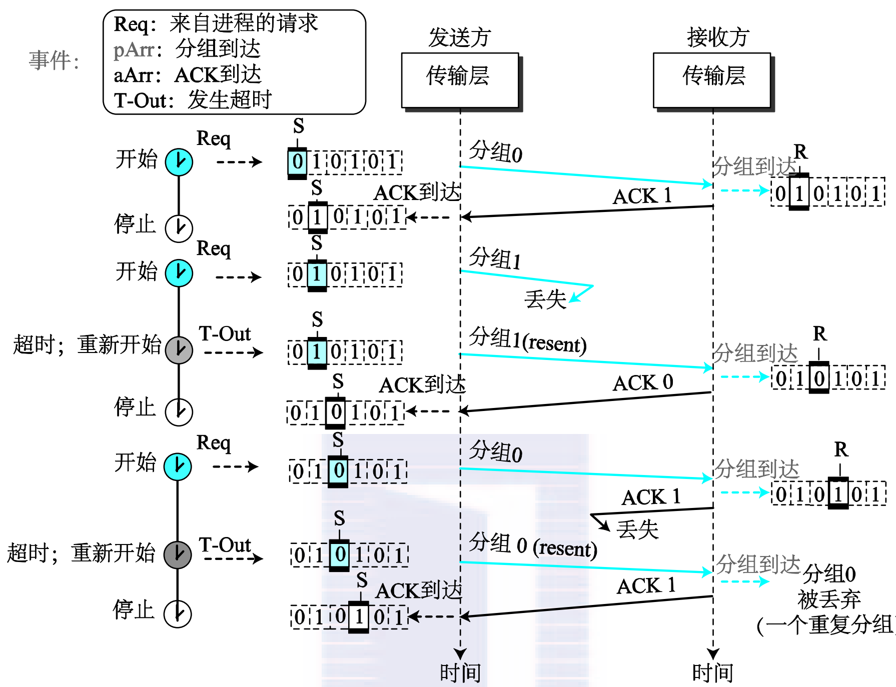


图 3-22　例 3.4 的流程图

效率

如果我们的信道又粗（thick）又长（long），那么停止–等待协议将非常低效。对于粗信道，我们的意思是信道有很大的带宽（高数据速率）；对于长信道，我们的意思是往返时间很长。这两者的乘积称为**带宽延迟乘积**（bandwidth-delay product）。我们可以把信道看做一条管道。带宽延迟乘积是以比特位为单位的管道容量。管道就在那里。如果不使用它，那么它就无效率。带宽延迟乘积可用来度量在等待来自接收方的确认信息时，发送方能够发送的位数。

例 3.5　假设在停止–等待系统中，线路的带宽是 1Mbps，1 比特需要花 20 毫秒完成往返。带宽延迟乘积是多少？如果系统数据分组长度是 1000 位，链路的利用率是多少？

解答

带宽延迟乘积是$(1 \times 10^6) \times (20 \times 10^{-3}) = 20\ 000$ 位。在数据从发送方到接收方以及确认返回的这段时间内，系统可以发送 20 000 位。然而，系统只发送了 1000 位。我们可以说链路利用率是 1000/20 000，或 5%。因此，在高带宽或大延时的链路中，停止–等待协议会浪费链路容量。

例 3.6　如果我们使用一个协议，在这个协议停止并担心确认之前，可以发送多达 15 个分组，那么例 3.5 中的链路利用百分率是多少？

解答

带宽延迟乘积仍是 20 000 位。系统在往返时间内可以发送多达 15 个分组或 15 000 位。这意味着利用率是 15 000/20 000 或 75%。当然，如果存在损坏的分组，利用率更小，这是因为分组需要重发。

流水线

在网络和其他领域中，在之前的任务结束前经常开始一个新任务。这称为**流水线**（pipeline）。

在停止-等待协议中没有流水线，因为发送端必须等待分组到达目的地，并且在下一个分组发送前接收到确认。然而，流水线用于我们下面两个协议，因为在发送方接收到关于前一个分组的反馈之前，就可以发送许多分组。如果与带宽延迟乘积相关的处于传输状态的比特位较多，那么流水线方式会提高传输的效率。

3.2.3　回退 *N* 帧协议

为了提高传输效率（充满管道），当发送端等待确认时，必须传输多个分组。换言之，当发送端等待确认时，我们需要让不止一个分组处于未完成状态，以此确保信道忙碌。在这一节中，我们讨论一个可以实现这个目标的协议；在下一节中，我们讨论第二个协议。第一个协议称为**回退 *N* 帧协议**（Go-Back-*N*，GBN，这个名字的道理之后就会明白）。回退 *N* 帧的关键是我们在接收到确认之前，可以发送多个分组，但是接收端只能缓冲一个分组。我们保存被发送分组的副本直到确认到达。图 3-23 给出了这个协议的框架。注意，很多数据分组以及确认可以同时处于信道中。

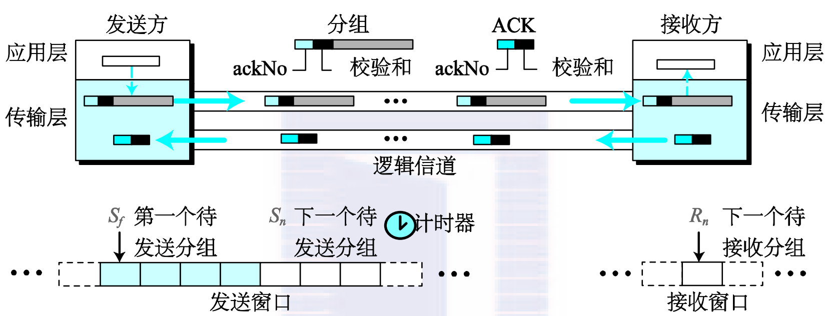


图 3-23　回退 *N* 帧协议

序号

如前所述，序号是模 2^m 的，这里 m 是序号字段的大小，单位是比特（位）。

确认号

这个协议中的确认号是累积的，并且定义了预期接收的下一个分组序号。例如，如果确认号（ackNo）是 7，这意味着序号在 6 以内的分组都已经安全完整到达，并且接收方等待序号为 7 的分组。

在回退 *N* 帧协议中，确认号是累积的并且定义了预期接收的下一个分组序号。

发送窗口

发送窗口是一个想象的盒子，它覆盖了处于运送途中的以及可以被发送的数据分组序号。在每个窗口位置，某些序号定义了已经被发送的分组；其他序号定义了可以被发送的分组。窗口最大为 2^m-1，我们之后讨论这其中的原因。在本章，我们令窗口大小固定为最大值，但是我们之后会看到，有些协议的窗口大小可能可以变化。图 3-24 给出回退 *N* 帧协议中大小为 7 的滑动窗口（$m = 3$）。

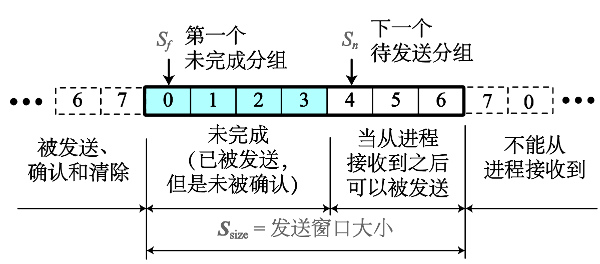


图 3-24　回退 *N* 帧发送窗口

在任何时候，发送窗口都可能将序号分成四部分。第一部分，窗口左侧，定义了已经确认的分组的序号。发送方不需要担心这些分组并且不需要保存它们的副本。第二部分，带灰度部分，定义了已经被发送的分组的序号，但是这些分组状态未知。发送方需要等待，从而发现这些分组究竟是已经被

接收还是丢失。我们把这些分组称为未完成（outstanding）分组。第三部分，浅灰部分，定义了可以发送的分组的序号；然而，相应数据还没有从应用层接收到。最后，第四部分，窗口右侧，定义了直到窗口滑动前都不能使用的序号。

窗口本身是一种抽象；三个变量定义了它任何时候的大小和位置。我们将这些变量称为 S_f（发送窗口，第一个未完成分组）、S_n（发送窗口，下一个待发送分组）以及 S_{size}（发送窗口，大小）。变量 S_f 定义了第一个（最旧的）未完成分组序号。变量 S_n 存储的序号将要被分配给下一个待发送的分组。最终，变量 S_{size} 定义了窗口大小，窗口大小在我们的协议中是固定不变的。

> 发送窗口是一种抽象概念，它定义了一个最大为 2^m-1 的想象的盒子，其中有三个变量，它们是 S_f、S_n 以及 S_{size}。

图 3-25 给出了当从另一端接收到确认时，发送窗口是如何向右滑动一到多个槽的。在图中，ackNo = 6 的确认到达。这意味着接收方正在等待序号为 6 的分组。

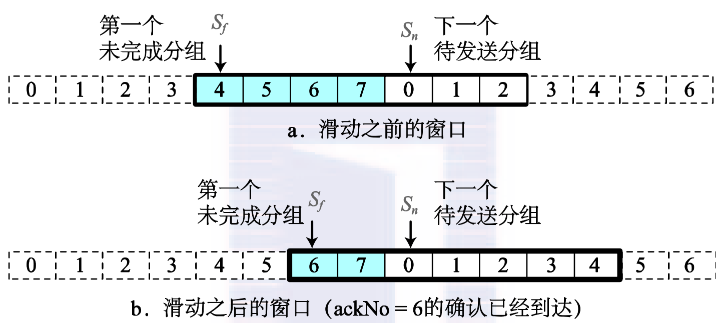


图 3-25 发送窗口的滑动

> 当 ackNo 大于等于 S_f 且小于 S_n（模运算）的无错 ACK 到达时，发送窗口可以滑动一个或多个槽。

接收窗口

接收窗口确保正确的数据分组被接收，并且确保正确的确认被发送。在回退 N 帧中，接收窗口的大小总是 1。接收方总是寻找特定分组是否到达。任何失序分组到达都会被丢弃并需要被重发。图 3-26 给出了接收窗口。注意，我们只需要一个变量，即 R_n（接收窗口，预期接收的下一个分组），来定义这种抽象窗口。窗口左侧的序号属于已经被接收和确认的分组；窗口右侧的序号定义了不能被接收的分组。任何序号在这两区域中的分组都被丢弃。只有序号符合 R_n 值的分组才能被接收和确认。接收窗口也滑动，但是一次只滑动一个槽。当正确的分组被接收时，窗口滑动 $R_n = (R_n + 1)$ modulo 2^m。

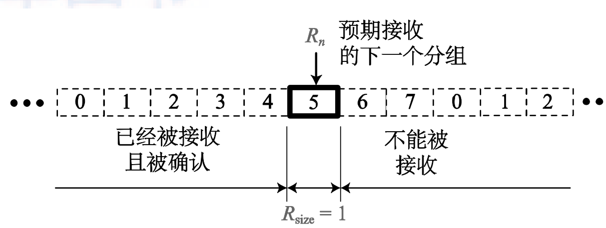


图 3-26 回退 N 帧接收窗口

> 接收窗口是一个抽象概念，它定义了一个最大为 1 的想象的盒子，其中只有一个变量 R_n。当正确分组到来时，窗口滑动；窗口每次只滑动一个槽。

计时器

尽管每个被发送分组都有计时器，但是在我们的协议中只使用一个计时器。原因是第一个未完成分组的计时器总是最先终止。当这个计时器终止时，我们重发所有未完成分组。

重发分组

当计时器终止时，发送方重发所有未完成分组。例如，假设发送方已经发送了分组 6（$S_n = 7$），但是唯一的计时器终止。如果 $S_f = 3$，这意味着分组 3、4、5 和 6 没有被确认；发送方回退并重发分组 3、4、5 和 6。这就是为什么称为回退 *N* 帧。一旦超时，机器回退 *N* 个位置并重发所有分组。

有限状态机

图 3-27 给出了 GBN 协议的有限状态机。

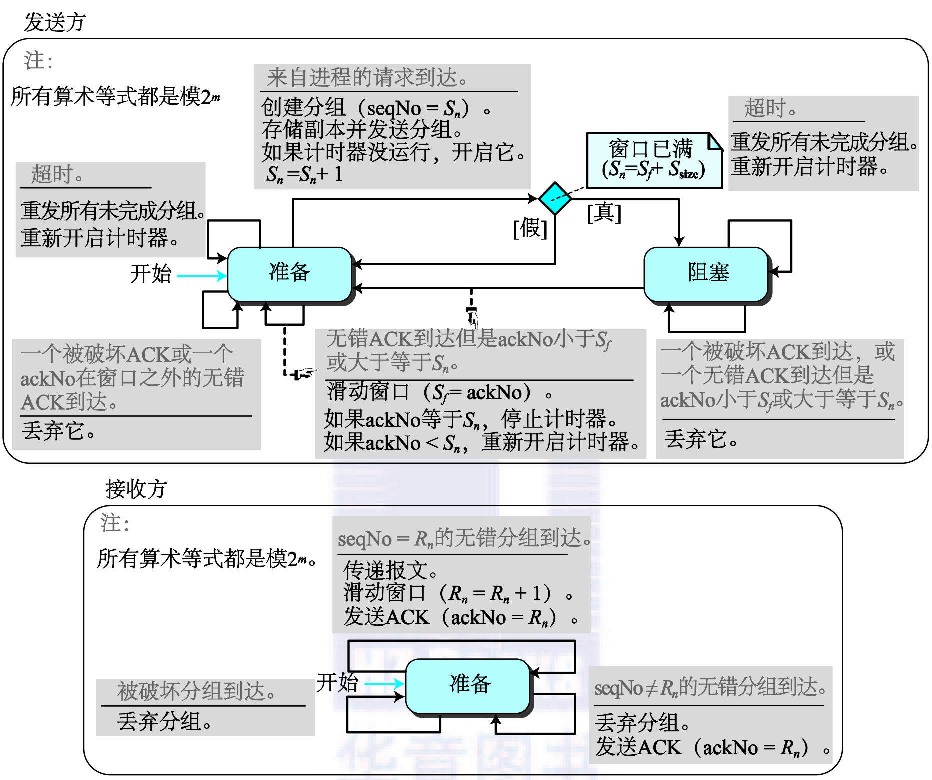


图 3-27　回退 *N* 帧协议有限状态机

发送方

发送方开始时处于准备状态，但是此后，它可能处于两种状态：准备（ready）或阻塞（blocking）。两个变量被初始化为 0（$S_f = S_n = 0$）。

- **准备状态**。当发送方处于准备状态，可能发生四个事件。

a. 如果请求来自应用层，发送方创建一个序号为 S_n 的分组。存储分组的副本，发送分组。如果计时器没有运行，发送方会开启唯一的计时器。S_n 的值增长，（$S_n = S_n + 1$）modulo 2^m。如果窗口已满，$S_n = (S_f + S_{size})$ modulo 2^m，发送方进入阻塞状态。

b. 如果无差错 ACK 到达，其 ackNo 与一个未完成分组有关，那么发送方滑动窗口（令 S_f = ackNo），并且如果所有未完成分组都被确认（ackNo = S_n），那么计时器停止。如果并不是所有未完成分组都被确认，那么计时器重新开启。

c. 如果一个被破坏 ACK 或 ackNo 与未完成分组无关的无错 ACK 到达，它就被丢弃。

d. 如果超时发生，发送方重发所有未完成分组并重新开启计时器。

- **阻塞状态**。在这种情况下可能发生三个事件：

a. 如果 ackNo 与一个未完成分组相关的无错 ACK 到达，那么发送方滑动窗口（令 S_f = ackNo），如果所有未完成分组被确认（ackNo = S_n），那么关闭计时器。如果所有未完成分组没有被确认，那

么重新开启计时器。之后，发送方进入准备状态。

b．如果一个被破坏 ACK 或 ackNo 与未完成分组无关的无错 ACK 到达，那么 ACK 被丢弃。

c．如果超时发生，发送方发送所有未完成分组并重新开启计时器。

接收方

接收方总是处于准备状态。唯一的变量 R_n 被初始化为 0。可能发生三个事件：

a．如果 seqNo = R_n 的无错分组到达，分组中的报文被传递到应用层。之后窗口滑动，R_n =（R_n + 1）modulo 2^m。最终 ackNo = R_n 的 ACK 被发送。

b．如果 seqNo 在窗口之外的无错分组到来，分组被丢弃，但是 ackNo = R_n 的 ACK 被发送。

c．如果被破坏分组到达，将被丢弃。

发送窗口大小

我们现在可以给出为什么发送窗口大小必须小于 2^m 了。举例来说，我们选择 m = 2，这意味着窗口大小可能是 2^m – 1 或 3。图 3-28 将大小为 3 的窗口与大小为 4 的窗口进行对比。如果窗口大小为 3（小于 2^m）并且所有三个确认都丢失，那么终止唯一的计时器并且重发所有三个分组。接收方现在期待分组 3，而不是分组 0，因此重复分组被正确丢弃。另一方面，如果窗口的大小是 4（等于 2^2）并且所有确认都丢失，发送方将会发送一个分组 0 的副本。然而，这次接收窗口期待接收的是分组 0（在下一轮），因此它接收分组 0，并不将其作为一个副本，但是作为下一轮的第一个分组。这是一个错误。这展示出发送窗口的大小必须小于 2^m。

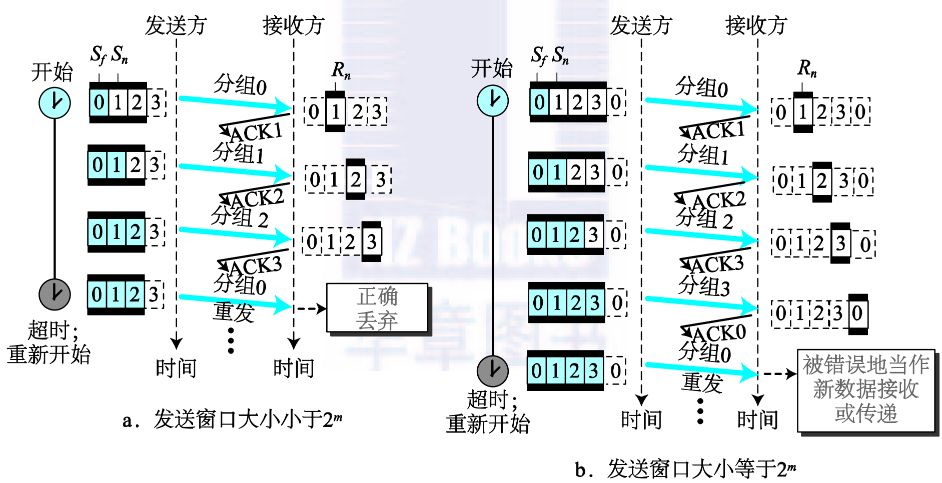


图 3-28 回退 N 帧发送窗口大小

在回退 N 帧协议中，发送窗口大小必须小于 2^m；接收窗口大小总是 1。

例 3.7 图 3-29 给出了回退 N 帧的例子。在这个例子中，转发信道是可靠的，但是反向是不可靠的。虽然没有数据分组丢失，但是一些 ACK 被延迟，还有一个 ACK 丢失。这个例子也给出了当确认延迟或丢失时，积累确认是如何起到帮助作用的。

在初始化之后，有一些发送方事件。请求事件被来自应用层的报文块触发；到达事件被来自网络层的 ACK 触发。这里没有超时事件，因为所有未完成分组在计时器超时之前都被确认。注意，尽管 ACK2 丢失，但是 ACK3 是积累的并且作为 ACK2 和 ACK3 进行服务。在接收端有四个事件。

例 3.8 图 3-30 给出当分组丢失时发生的事情。分组 0、1、2 和 3 被发送。然而，分组 1 丢失。接收方接收分组 2 和 3，但是由于它们是失序的（分组 1 是预期到达的分组），因此被丢弃。当接收方接收到分组 2 和 3 时，它发送 ACK1 来表示它预期接收分组 1。然而，这些 ACK 对发送方是无用的，因为 ackNo 等于 S_f，而不大于 S_f。因此发送方丢弃它们。当超时事件发生时，发送方重新

发送分组 1、2 和 3，它们被确认。

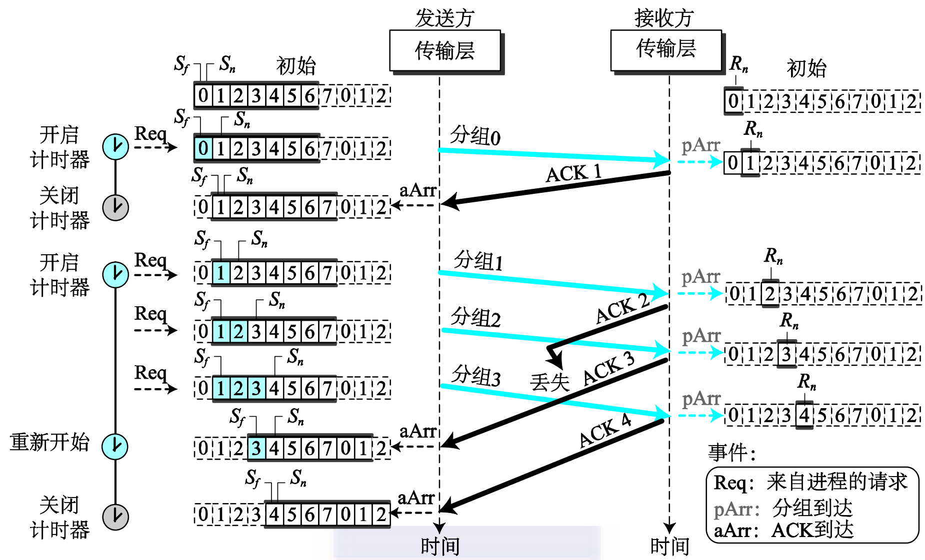


图 3-29　例 3.7 的流程图

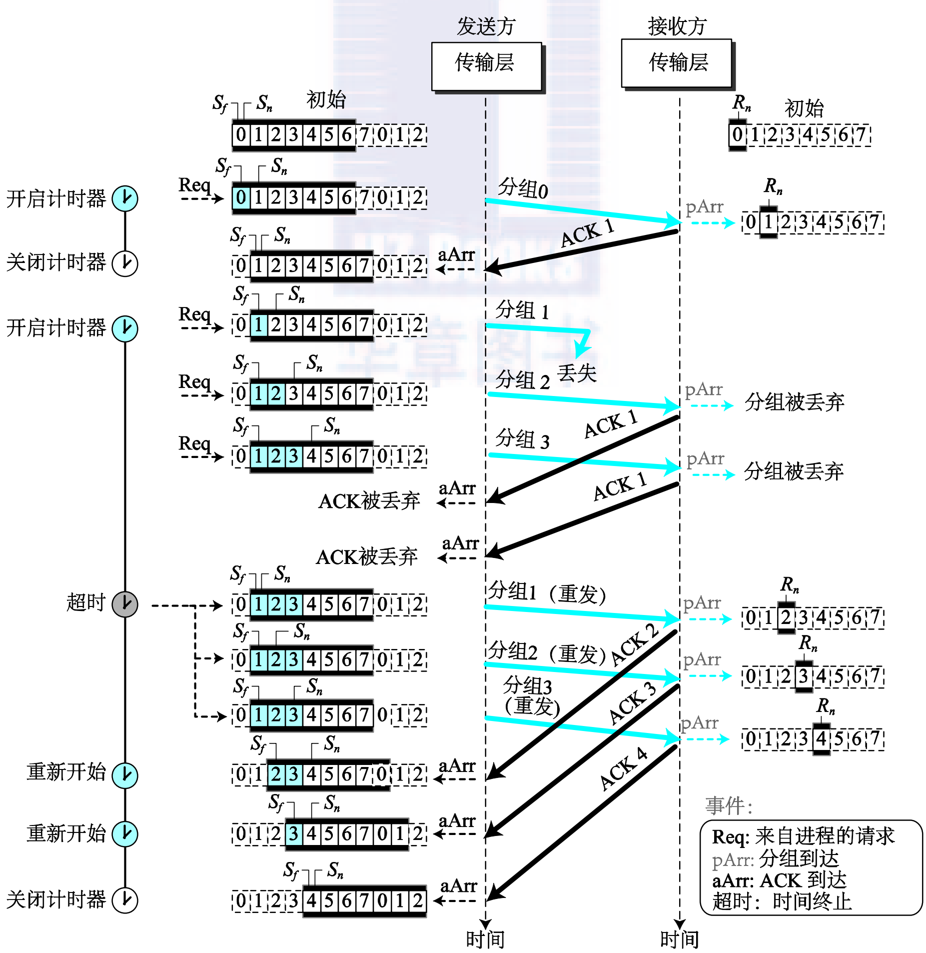


图 3-30　例 3.8 的流程图

回退 *N* 帧与停止等待

读者可能发现回退 N 帧协议与停止-等待协议有一些相似。停止-等待协议实际上是一种回退 N 帧协议，这种回退 N 帧协议只有两个序号且发送窗口大小为 1。换言之，$m = 1$ 且 $2^m - 1 = 1$。在回退 N 帧中，我们说计算是模 2^m 进行的；在停止-等待协议中是模 2，这与 $m = 1$ 时的 2^m 相同。

3.2.4　选择性重复协议

回退 N 帧协议简化了接收方的进程。接收方只记录一个变量，没有必要缓冲失序分组；它们被简单地丢弃。然而，如果下层网络层丢失很多分组，那么这个协议是低效的。每当一个分组丢失或被破坏，发送方要重新发送所有未完成分组，即使有些失序分组已经被安全完整地接收了。如果网络层由于网络拥塞，丢失了很多分组，那么重发所有这些未完成分组将会使得拥塞更严重，最终更多的分组丢失。这具有雪崩效应，可能导致网络全部瘫痪。

另一个协议，称为**选择性重复协议**（Selective-Repeat（SR）protocol），已经被设计出来，正如其名字所示，只是选择性重发分组，即那些确实丢失的分组。这个协议的框架如图 3-31 所示。

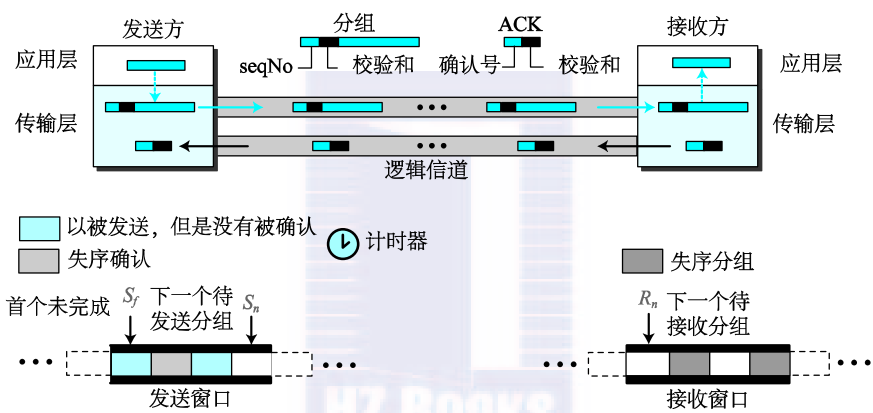


图 3-31　选择性重复框架

窗口

选择性重复协议也使用两个窗口：一个发送窗口和一个接收窗口。然而，这些窗口与回退 N 帧中的不同。首先，发送窗口的最大值更小；它是 2^{m-1}。这里的原因我们稍后讨论。第二，接收窗口和发送窗口大小一致。

发送窗口最大为 2^{m-1}。例如，如果 $m = 4$，序号从 0 到 15，但是窗口最大值仅仅是 8（在回退 N 帧协议中是 15）。我们在图 3-32 中给出选择性重复发送窗口来着重强调窗口大小。

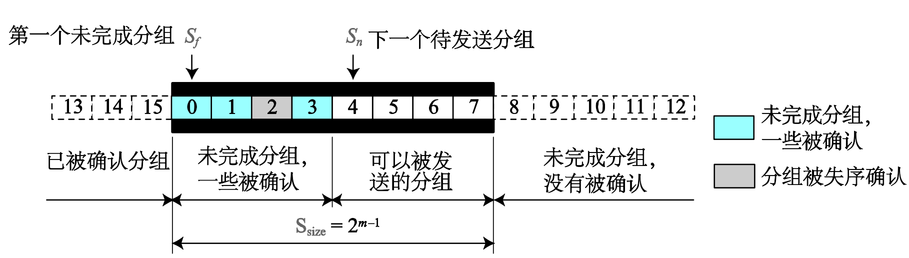


图 3-32　选择性重复协议发送窗口

选择性重复接收窗口与回退 N 帧中的接收窗口完全不同。接收窗口的大小和发送窗口等大（最大 2^{m-1}）。选择性重复协议允许和接收窗口一样多的分组失序到来并被存储，直到有一组连续分组被传递到应用层。因为发送窗口和接收窗口的大小是相同的，在发送窗口的所有分组可以失序到达

并被存储，直到它们可以被传递。然而，我们需要强调的是，在可靠协议中，接收方从不向应用层传递失序分组。图 3-33 给出选择性重复中的接收窗口。那些带阴影的窗口内的槽定义了失序到达的分组，并且在传输到应用层之前正在等待早先发送的分组。

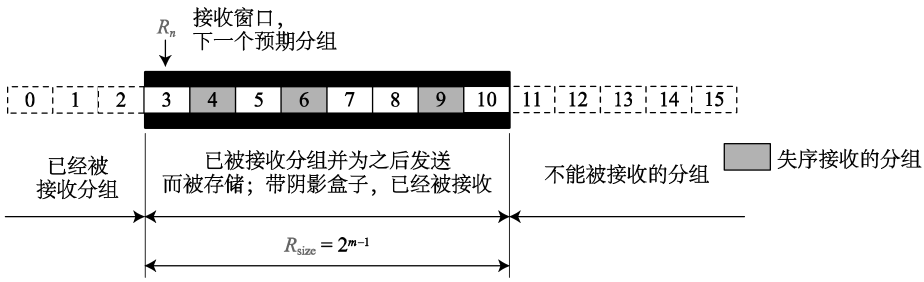


图 3-33　选择性重复协议接收窗口

计时器

理论上讲，选择性重复为每个未完成分组使用一个计时器。当一个计时器终止，只有一个相应分组被发送。换言之，GBN（回退 *N* 帧）将未完成分组看做一个组；SR（选择性重复）将它们单独处理。然而，绝大多数实现了 SR 的传输层协议只使用一个计时器。出于这个原因，我们只使用一个计时器。

确认

这两个协议之间还有一点不同。在 GBN 中 ackNo 是累积的；它定义了下一个预期分组的序号，确认了之前的分组都安全完整到达。在 SR 中确认的语义是不同的。在 SR 中，ackNo 定义了被安全完整接收的一个分组；对其他分组没有反馈信息。

在选择性重复协议中，确认号定义了已被接收的无错分组的序号。

例 3.9　假设一个发送方发送 6 个分组：分组 0、1、2、3、4 和 5。发送方接收 ackNo = 3 的 ACK。如果系统使用 GBN 或 SR，那么这该如何解释呢？

解答

如果系统使用 GBN，这意味着分组 0、1 和 2 已经被无误接收，并且接收方正在期待分组 3。如果系统正在使用 SR，这意味着分组 3 已经被无误接收；ACK 并没有涉及其他分组的信息。

有限状态机

图 3-34 给出选择性重复协议的有限状态机。它与 GBN 类似，但有一些不同。

发送方

发送方开始时处于准备状态，但是此后，它可能处于两种状态：准备或阻塞。以下列出了事件以及每个状态的相关动作。

- **准备状态**。当发送方处于准备状态，可能发生四个事件：

a．如果请求来自应用层，发送方创建一个序号为 S_n 的分组。分组的副本被存储，分组被发送。如果计时器没有在运行，发送方开启计时器。现在 S_n 值开始增长，（$S_n = S_n + 1$）modulo 2^m。如果窗口已满，$S_n =$（$S_f + S_{size}$）modulo 2^m，发送方进入阻塞状态。

b．如果无差错 ACK 到达，其 ackNo 与一个未完成分组有关，分组被标记为未确认。如果 ackNo = S_f，那么窗口向右方滑动，直到 S_f 指向第一个未确认分组（所有连续的已被确认分组现在处于窗口之外）。如果存在未完成分组，重新开启计时器；否则，停止计时器。

c．如果一个被破坏分组或 ackNo 与未完成分组无关的无错分组到达，它就被丢弃。

d．如果超时发生，发送方重发所有未完成分组并重新开启计时器。

- **阻塞状态**。在这种情况下可能发生三个事件：

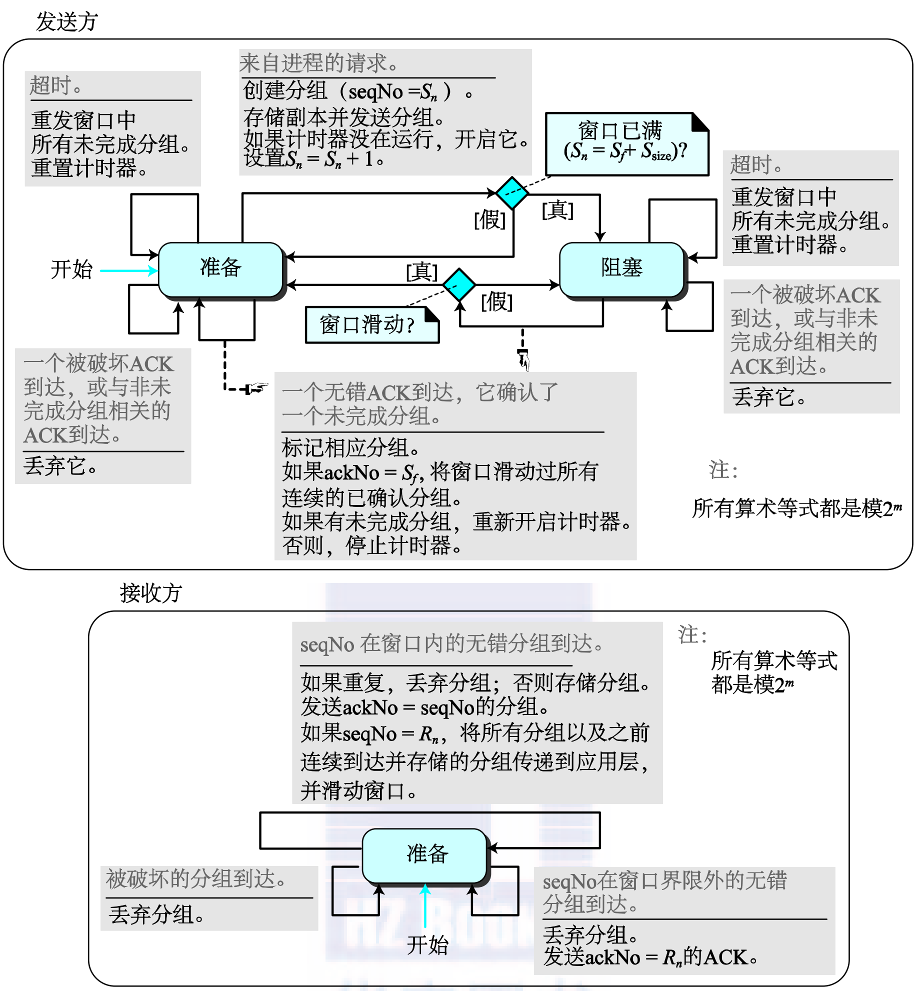


图 3-34　选择性重复协议有限状态机

a．如果 ackNo 与一个未完成分组相关的无错 ACK 到达，那么这个分组被标记为已确认。此外，如果 ackNo = S_f，那么窗口向右滑动，直到 S_f 指向第一个未确认分组（所有连续的已被确认分组现在处于窗口之外）。如果窗口已经滑动，发送方进入准备状态。

b．如果一个被破坏 ACK 或 ackNo 与未完成分组无关的无错 ACK 到达，那么 ACK 被丢弃。

c．如果超时发生，发送方发送所有未完成分组并重新开启计时器。

接收方

接收方总是处于准备状态。可能发生三个事件：

a．如果 seqNo 在窗口内的无错分组到达，分组被存储并且 ackNo = seqNo 的 ACK 被发送。此外，如果 seqNo = R_n，那么分组以及之前连续到达的分组被传递到应用层，并且窗口滑动使得 R_n 指向第一个空槽。

b．如果 seqNo 在窗口之外的无错分组到达，分组被丢弃，但是 ackNo = R_n 的 ACK 被返回到发送方。如果那些与 seqNo < R_n 分组相关的 ACK 丢失，那么窗口需要滑动。

c．如果被破坏分组到达，它被丢弃。

例 3.10　这个例子与例 3.8（图 3-30）相似，其中分组 1 丢失。我们给出选择性重复协议在这种情况下如何工作。图 3-35 给出了具体场景。

在发送方，分组 0 被传输并且被确认。分组 1 丢失。分组 2 和 3 失序到达并被确认。当计时器超时，分组 1（唯一未被确认的分组）被重发并被确认。之后，发送窗口滑动。

在接收端我们需要将分组接收和传递分组到应用层区分开。在第二次到达事件中，分组 2 到达了且被存储、被标记（阴影槽），但是它不能被传递，因为分组 1 丢失了。在下一个到达事件中，分组 3 到达了且被标记、被存储，但是分组仍然不能被传递。只有在最后一次到达事件中，当最终分组 1 的副本到达了，分组 1、2 和 3 才能被传输到应用层。将分组传输到应用层有两个条件：第一，一组连续的分组必须已经到达。第二，这一组分组开始于窗口的起始端。在最后一次到达事件发生之后，有三个分组并且第一个分组开始于窗口起始端。关键是在于可靠传输层保证按序传输分组。

窗口大小

我们现在给出为什么发送窗口和接收窗口最多为 2^m 的一半。例如，我们选择 $m = 2$，这意味着窗口大小为 $2^m/2$ 或 $2^{(m-1)} = 2$。图 3-36 将大小为 2 的窗口与大小为 3 的窗口进行比较。

如果窗口大小为 2，并且所有确认丢失，那么分组 0 的计时器超时且分组 0 被重发，因此这个重复分组被正确地丢弃（序号 0 不在窗口内）。当窗口大小为 3，并且所有确认丢失，那么发送方发送分组 0 的副本。然而，这次，接收方窗口期待接收分组 0（0 是窗口的一部分），因此它接收了分组 0，并且不把它看做一个重复分组，但是作为下一个循环中的分组。这明显是一个错误。

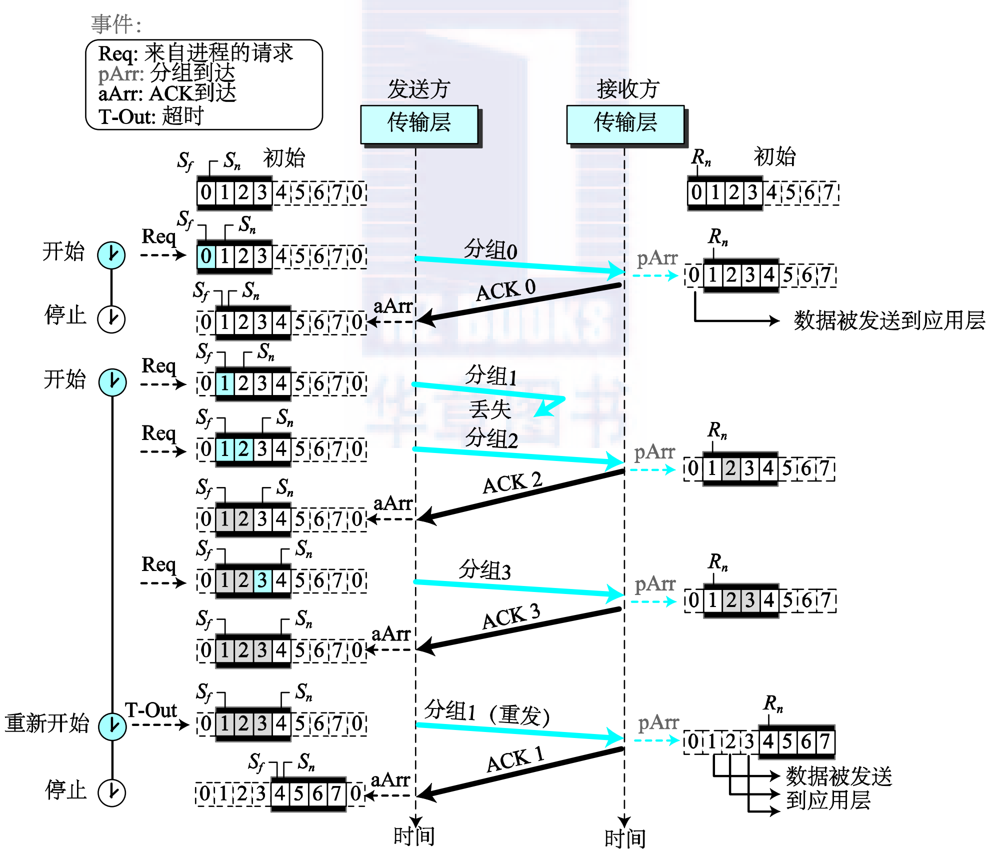


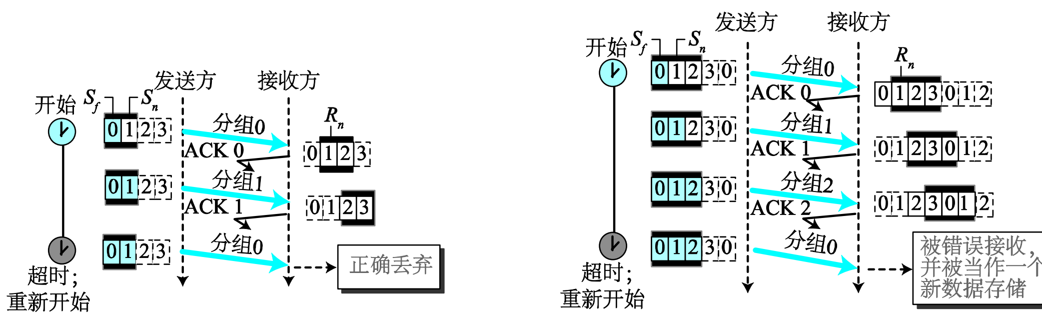
图 3-35　例 3.10 流程图

在选择性重复中，发送方和接收方窗口的大小最多为 2^m 的一半。

3.2.5　双向协议：捎带

我们在本节讨论的四个协议都是单向的：数据分组只沿着一个方向流动并且确认也是按一个方

向传递的。在现实生活中，数据分组通常是双向流动的：从客户到服务器以及从服务器到客户。这意味着确认也需要沿着两个方向流动。一种称为**捎带**（piggybacking）的技术被用来提高双向协议的效率。当一个分组携带数据从 A 到 B 时，它也携带了确认反馈，这些信息确认了来自 B 的分组已到达；当一个分组携带数据从 B 到 A 时，它也携带了确认反馈，这些信息确认了来自 A 的分组已到达。



a．发送和接收窗口大小为 2^{m-1}　　b．发送和接收窗口大小大于 2^{m-1}

图 3-36　选择性重复，窗口大小

图 3-37 给出了使用捎带实现了双向功能的 GBN 协议。客户和服务器各使用两个独立窗口：发送和接收。

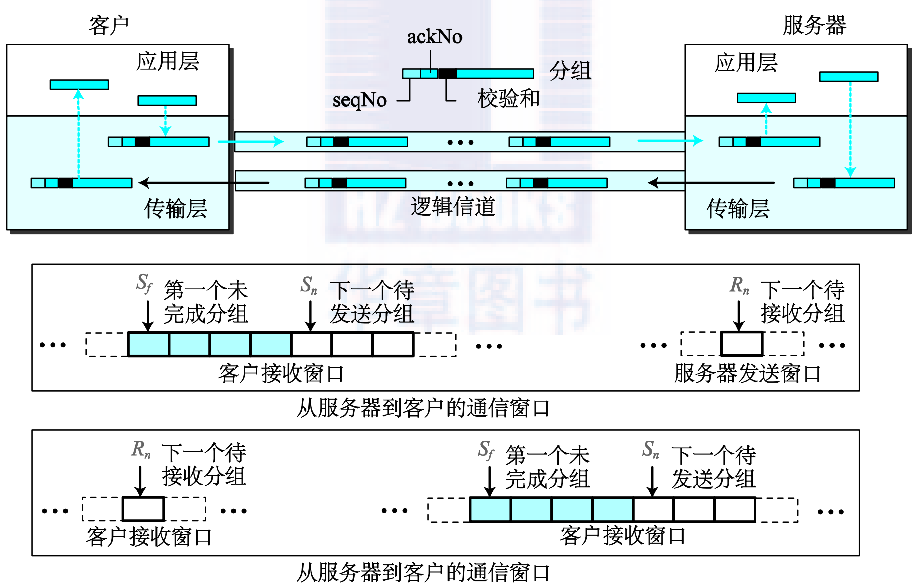


图 3-37　回退 N 帧中的捎带设计

3.2.6　因特网传输层协议

在讨论传输层的一般原则之后，我们在下面两节专注于因特网的传输层协议。尽管因特网使用很多传输层协议，但是我们在本章只讨论两个，如图 3-38 所示。

图 3-38 中给出了 UDP 和 TCP 这两个传输层协议与其他协议的关系，以及 TCP/IP 协议簇的层次。这些协议位于应用层和网络层之间，是应用程序和网络操作的中间媒介。

UDP 是不可靠的无连接传输层协议，由于在应用中简单高效而被使用，在那些应用中差错控制由应用层进程提供。TCP 是可靠的面向连接协议，可用于可靠性重要的任何应用。也有其他应

用层协议比如 SCTP，我们会在其他章节讨论它们。

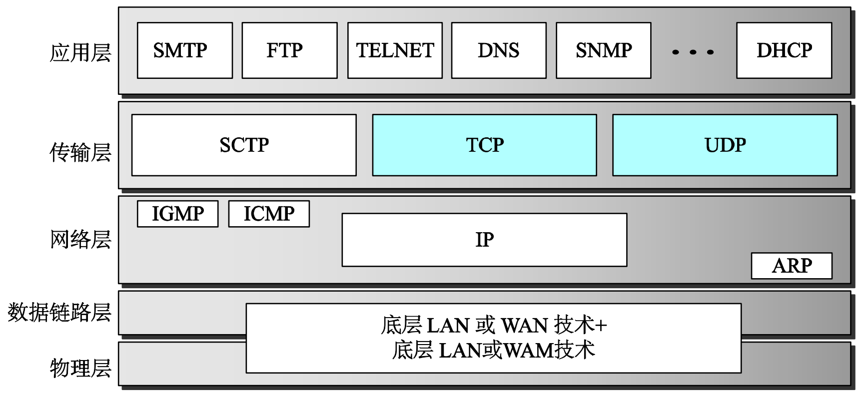


图 3-38　在 TCP/IP 协议簇中传输层协议的位置

如前所述，传输层协议通常有很多责任。一个是创建进程到进程通信；这些协议使用端口号来完成这项责任（见表 3-1）。

表 3-1　UDP 和 TCP 使用的熟知端口

端　　口	协　　议	UDP	TCP	说　　明
7	**Echo**	√		将接收到的数据报回送到发送方
9	**Discard**	√		丢弃接收到的任何数据报
11	**Users**	√	√	活跃的用户
13	**Daytime**	√	√	返回日期和时间
17	**Quote**	√	√	返回每日引用
19	**Chargen**	√	√	返回一字符串
20,21	**FTP**		√	文件传输协议
23	**TELNET**		√	终端网络
25	**SMTP**		√	简单邮件传输协议
53	**DNS**	√	√	域名服务
67	**DHCP**	√	√	动态主机设置协议
69	**TFTP**	√		简单文件传输协议
80	**HTTP**		√	超文本传输协议
111	**RPC**	√	√	远程过程调用
123	**NTP**	√	√	网络时间协议
161 162	**SNMP**		√	简单网络管理协议

3.3　用户数据报协议

用户数据报协议（User Datagram Protocol，UDP）是无连接不可靠传输层协议。它不提供主机到主机通信，它除了提供进程到进程之间的通信之外，就没有给 IP 服务增加任何东西。此外，它进行非常有限的差错检验。如果 UDP 功能是如此之差，那么为什么进程还要使用它？它有缺点也有优点。UDP 是一个非常简单的协议，开销最小。如果一个进程想发送很短的报文，而且不在意可靠性，就可以使用 UDP。使用 UDP 发送一个很短的报文，在发送方和接收方之间的交互要比使用 TCP 时少得多。我们在本节最后讨论一些使用 UDP 的一些应用。

3.3.1　用户数据报

UDP 分组称为**用户数据报**（user datagram），有 8 字节的固定头部，这个头部由 4 个字段组成，每个字段 2 字节（16 位）。图 3-39 说明了用户数据报的格式。头两个字段定义了源和目的端口号。第三个字段定义了用户数据报的总长，即头部加数据的长度。16 位可以定义的总长度范围是 0 到 65 535。然而，总长度需要更小一些，这是因为 UDP 数据报存储在总长度为 65 535 的 IP 数据报中。最后一个字段可以携带可选校验和（稍后解释）。

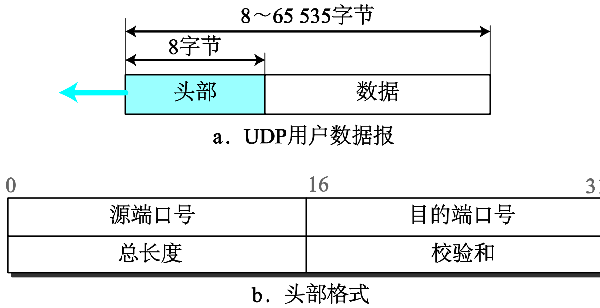


图 3-39　用户数据报格式

例 3.11　以下是十六进制格式的 UDP 头部内容。

CB84000D001C001C

a．源端口号是多少？
b．目的端口号是多少？
c．用户数据报总长度是多少？
d．数据长度是多少？
e．分组是从客户发向服务器的还是相反方向的？
f．客户进程是什么？

解答

a．源端口号是头 4 位十六进制数字（CB84）$_{16}$，这意味着源端口号是 52100。
b．目的端口号是第二组 4 位十六进制数字（000D）$_{16}$，这意味着目的端口号是 13。
c．第三组 4 位十六进制数字（001C）$_{16}$ 定义了整个 UDP 分组的长度，长度为 28 字节。
d．数据的长度是整个分组长度减去头部长度，即 28–8 = 20 字节。
e．由于目的端口号是 13（熟知端口号），分组是从客户发送到服务器。
f．客户进程是 Daytime（见表 3-1）。

3.3.2　UDP 服务

我们早先讨论过传输层协议提供的一般服务。在本节，我们讨论 UDP 提供的一般服务。

进程到进程的通信

UDP 使用套接字地址提供进程到进程通信，这是 IP 地址和端口号的组合。

无连接服务

如前所述，UDP 提供无连接服务。这就是表示 UDP 发送出去的每一个用户数据报都是一个独立的数据报。不同的用户数据报之间没有关系，即使它们都是来自相同的源进程并发送到相同的目的程序。用户数据报不进行编号。此外，也没有像 TCP 协议那样的连接建立和连接终止，这就表示每一个用户数据报可以沿着不同的路径传递。

无连接的一个结果就是使用 UDP 的进程不能够向 UDP 发送数据流，并期望它将这个数据流分割成许多不同的相关联的用户数据报。相反，每一个请求必须足够小，使其能够装入用户数据报中，只有那些发送短报文的进程才应当使用 UDP。短报文小于 65 507 字节（65 535 减去 UDP 头部的 8 字节再减去 IP 头部的 20 字节）。

流量控制

UDP 是一个非常简单的协议。它没有流量控制（flow control），因而也没有窗口机制。如果到

来的报文太多时，接收方可能会溢出。缺乏流量控制意味着如果需要的话，使用 UDP 的进程应该提供这个服务。

差错控制

除校验和外，UDP 也没有差错控制（error control）机制，这就表示发送方不知道报文是丢失还是重传。当接收方使用校验和检测出差错时，它就悄悄地将此用户数据报丢弃。缺乏差错控制意味着如果需要的话，使用 UDP 的进程应该提供这个服务。

校验和

我们在第 5 章讨论检验和，以及它的计算。UDP 校验和包含三部分：伪头部、UDP 头部和从应用层来的数据。伪头部（psedoheader）是 IP 分组的头部的一部分（第 4 章讨论），其中有些字段要填入 0，用户数据报分装在 IP 分组中（见图 3-40）。

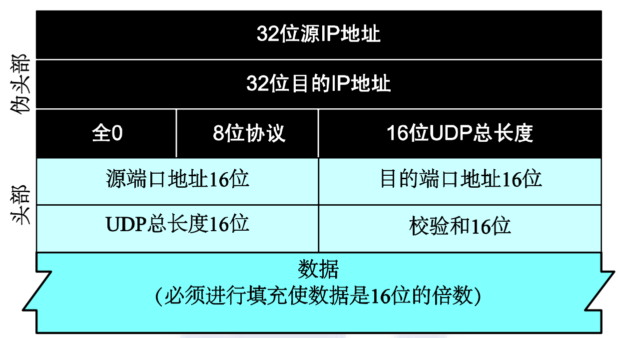


图 3-40　用于校验和计算的伪头部

如果校验和不包括伪头部，用户数据报也可能是安全完整地到达。但是，如果 IP 头部受到损坏，那么它可能被提交到错误的主机。

增加协议字段可确保这个分组是属于 UDP，而不是属于其他传输层协议。我们在后面将会看到，如果一个进程既可用 UDP 又可用 TCP，则端口号可以是相同的。UDP 的协议字段值是 17。如果在传输过程中这个值改变了，在接收端计算校验和时就可检测出来，UDP 就可丢弃这个分组。这样就不会传递给错误的协议。

可选校验和

UDP 分组的发送方可以选择不计算校验和。这种情况下，在发送前，校验和字段就全填入 0。在发送方决定计算校验和的情况下，如果碰巧结果全是 0，那么在发送前校验和全改为 1。换言之，发送方填充两次校验和。注意，这不会产生混淆，因为校验和的值在正常情况下不会全为 1（见例 3.12）。

例 3.12　在以下假想情况下校验和的数值是多少？

a. 发送方决定不包含校验和。

b. 发送方决定包含校验和，但是数值全为 1。

c. 发送方决定包含校验和，但是数值全为 0。

解答

a. 校验和字段全为 0 表示未计算校验和。

b. 当发送方填充校验和时，结果是全 0；发送方在发送前再次填充。数值为全 1。需要进行第二次填充操作来防止与 a 的情况混淆。

c. 这种情况不会再次发生，因为这暗示了计算中涉及的每个项目的数值为 0，这是不可能的；伪头部中的某些字段具有非零值。

拥塞控制

由于 UDP 是无连接协议，它不提供拥塞控制。UDP 假设被发送的分组很小且零星，不会在网络中造成拥塞。今天当 UDP 被用做音频和视频的交互实时传输时，这个假设可能对也可能不对。

封装和解封装

要将报文从一个进程发送到另一个进程时，UDP 协议就要对报文进行封装和解封装。

排队

我们已经讨论过端口，但是没有讨论端口的实际实现。在 UDP 中，队列是与端口联系在一起的。

在客户端，当进程启动时，它从操作系统请求一个端口号。有些实现是创建一个入队列和一个出队列与每一个进程相关联。而有些实现只创建与每一个进程相关的入队列。

多路复用与多路分解

在运行 TCP/IP 协议簇的主机上只有一个 UDP，但可能有多个想使用 UDP 服务的进程。处理这种情况，UDP 采用多路复用和多路分解。

UDP 和通用简单协议比较

我们可以将 UDP 与之前讨论的无连接简单协议进行比较。唯一的区别就是 UDP 提供可选校验和来在接收端发现被破坏分组。如果校验和被加入分组，接收 UDP 可以检测分组，如果分组被破坏可以丢弃它。然而，没有反馈被发向发送方。

> UDP 是我们之前讨论的无连接简单协议的一个例子，区别在于它为差错检测加入了可选校验和。

3.3.3　UDP 应用

尽管 UDP 不满足我们之前讨论的可靠传输层协议标准，但是，UDP 更适合某些应用。原因是其他某些服务可能有副作用，这些副作用或许是不可接受的或许是不称心的。一位应用设计师有时需要折中来得到最佳情况。例如，在日常生活中，我们都知道一日递送包裹比三日递送要贵。尽管时间和代价在递送包裹中都是想要获取的特性，但是它们是彼此矛盾的。我们需要选择最佳值。

在这一节，我们首先讨论 UDP 的一些特性，这些特性是当某人设计应用程序时需要的，然后，我们给出一些典型应用。

UDP 特性

我们简要讨论 UDP 的一些特性以及它们的优势和劣势。

无连接服务

如前所述，UDP 是无连接协议。同一个应用程序发送的 UDP 分组之间是独立的。这个特性可以看做是优势也可以看做是劣势，这要取决于应用要求了。例如，如果一个客户应用需要向服务器发送一个短的请求并接收一个短的响应，那么这就是优势。如果请求和响应各自可以填充进一个数据报，那么无连接服务可能更可取。在这种情况下，建立和关闭连接的开销可能很可观。在面向连接服务中，要达到以上目标，至少需要在客户和服务器之间交换 9 个分组；在无连接服务中只需要交换 2 个分组。无连接服务提供了更小的延迟；面向连接服务造成了更多的延迟。如果延迟是应用的重要问题，那么无连接服务更可取。

例 3.13　一种客户-服务器应用如 DNS（见第 2 章），它使用 UDP 服务，因为客户需要向服务器发送一个短的请求，并从服务器接收快速响应。请求和响应可以填充进一个用户数据报。由于在每个方向上只交换一个报文，因此无连接特性不是问题；客户或服务器不担心报文会失序传递。

例 3.14　一种客户-服务器应用如 SMTP（见第 2 章），它在电子邮件中使用，它不使用 UDP 服务，因为用户可能发送较长的电子邮件报文，邮件可能包含多媒体（图片、音频或视频）。如果

应用使用 UDP 且报文不能填充进一个用户数据报，那么报文必须被应用分割成不同的数据报。在这里，无连接服务可能产生问题。用户数据报可能失序到达并被传送到接收方应用。接收方应用可能无法重排这些片段。这意味着无连接服务对发送较长报文的应用来说有缺点。在 SMTP 中，当用户发送报文时，用户不期待很快收到响应（有时不需要响应）。这意味着面向连接服务中固有的额外延迟对 SMTP 来说不是至关重要的。

缺乏差错控制

UDP 不提供差错控制；它提供的是不可靠服务。绝大多数应用期待从传输层协议中得到可靠服务。尽管可靠服务是人们想要的，但是它可能有一些副作用，这些副作用对某些应用来说不可接受。当一个传输层提供可靠服务时，如果报文的一部分丢失或被破坏，它就需要被重传。这意味着接收方传输层不能向应用立即传送那一部分；在传向应用层的不同报文部分间会有不一致的延迟。对于某些应用天生就根本注意不到这些不一致的延迟，而对于有些应用这些延迟却是至关重要的。

例 3.15　假设我们正在从因特网下载一个非常大的文本文件。我们肯定需要使用提供可靠服务的传输层。当我们打开文件的时候，我们不想看到部分文件丢失或被破坏。每个报文之间的延迟对我们来说不是首要担心的事情；我们在整个文件构建好之前一直等待，然后查看它。在这种情况下，UDP 不是一个合适的传输层协议。

例 3.16　假设我们正在使用一个实时交互应用，例如 Skype。音频和视频被分割成帧并且一个接一个地发送。如果传输层应该重传某些被破坏或丢失的帧，那么整个传输的同步性就会丧失。观众会突然看到空白屏幕并且需要等待，直到第二个传输到达。这是不可容忍的。然而，如果屏幕的每个小部分都使用一个用户数据报传送，那么接收 UDP 可以轻易地忽略被破坏或丢失的分组，并将其余分组传递到应用程序。屏幕的那部分会空白很短一段时间，而绝大多数观众都不会注意到。

缺乏拥塞控制

UDP 不提供拥塞控制。然而，在倾向于出错的网络中 UDP 没有创建额外的通信量。TCP 可能多次重发一个分组，因此这个行为促使拥塞发生或者使得拥塞状况加重。因此，在某些情况下，当拥塞是一个大问题时，UDP 中缺乏差错控制可以看做是一个优势。

典型应用

下面给出了一些典型应用，与 TCP 服务相比，它们从 UDP 服务中的获益更多。

- UDP 适合于这样的进程：它需要简单的请求-响应通信，而较少考虑流量控制和差错控制。对于需要传送成块数据的进程（如 FTP）则通常不使用 UDP（见第 2 章）。
- UDP 适用于具有内部流量控制和差错控制机制的进程。例如，简单文件传输协议（TFTP）的进程就包含流量控制和差错控制。它可很容易地使用 UDP。
- 对多播来说，UDP 是一个合适的传输协议。多播能力已嵌入到 UDP 软件中，但没有嵌入到 TCP 软件中。
- UDP 可用于管理进程，如 SNMP（见第 9 章）。
- UDP 可用于某些路由选择更新协议，如路由选择信息协议（RIP）（见第 4 章）。
- UDP 通常用于交互实时应用，这些应用不能忍受接收报文之间的不一致延迟（见第 8 章）。

3.4　传输控制协议

传输控制协议（Transmission Control Protocol，TCP）是一个面向连接可靠的协议。TCP 显式定义了连接建立、数据传输以及连接拆除阶段来提供面向连接服务。TCP 使用 GBN 和 SR 协议的组合来提供可靠性。为了实现这个目的，TCP 使用校验和（为差错发现）、丢失或被破坏分组重传、累积和选择确认以及计时器。在这一节，我们首先讨论 TCP 提供的服务；之后我们详细讨论 TCP 的特性。TCP 是因特网中最常见的传输层协议。

3.4.1　TCP 服务

在详细讨论 TCP 之前，让我们解释 TCP 向应用层提供的服务。

进程到进程的通信

像 UDP 一样，TCP 通过使用端口号来提供进程到进程通信。在表 3-1 中给出了 TCP 使用的一些端口号。

流传递服务

与 UDP 不同，TCP 是一个面向流的协议。在 UDP 中，进程发送一些具有预先规定边界的报文给 UDP 进行传递。UDP 将它自己的头部添加到这些报文中并传递到 IP 层进行传输。来自进程的每一个报文称为一个用户数据报，最后变成一个 IP 数据报。IP 和 UDP 都不认识这些数据之间的关系。

另一方面，TCP 允许发送进程以字节流形式传递数据，并且接收进程也以字节流形式接收数据。TCP 建立一种环境，在这种环境中，两个进程好像由一个假想的“管道”连接，这个管道通过因特网传送这些进程的数据。这种假想的环境如图 3-41 所示。发送进程产生（写入）字节流，而接收进程消费（读出）这些字节流。

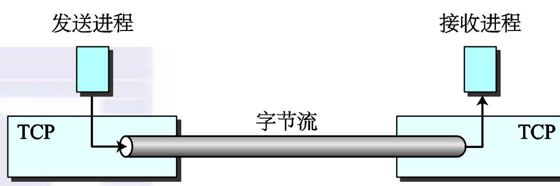


图 3-41　字节流传递

发送和接收缓冲区

因为发送和接收进程可能以不同的速度写入和读出数据，所以 TCP 需要用于存储的缓冲区。每一个方向都存在一个缓冲区：发送缓冲区和接收缓冲区。稍后我们会看到，这些缓冲区也用于 TCP 流量和差错控制机制。实现缓冲区的一种方法是使用以一字节为存储单元的循环数组，如图 3-42 所示。为了简化，我们只画出了两个缓冲区，每个缓冲区 20 个字节。通常情况下，缓冲区是数百甚至数千个字节，这取决于实现方法。这里给出的缓冲区是大小相同的，实际上并非总是如此。

HZ BOOKS 华章图书

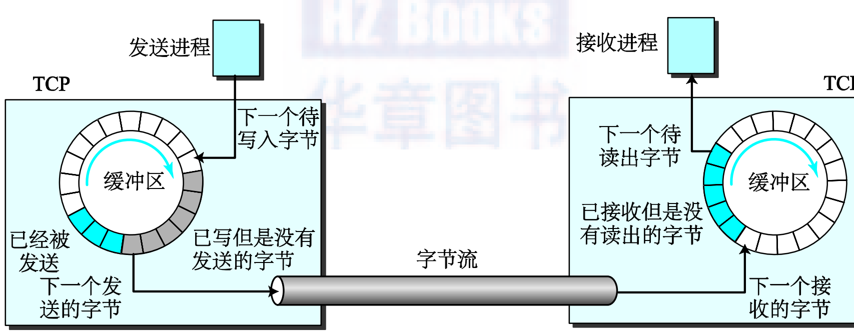


图 3-42　发送与接收缓冲区

图 3-42 表示了在一个方向上数据的移动。在发送端，缓冲区有三种类型的存储单元。白色的部分是空存储单元，可以由发送进程（生产者）填充。灰色的部分用于保存已经发送但还没有得到确认的字节。TCP 在缓冲区中保留这些字节，直到收到确认为止。灰色缓冲区是将要由 TCP 发送的字节。但是，在本章的后面将会看到，TCP 可能只发送灰色部分。这可能是由于接收进程缓慢或者网络中可能发生的拥塞造成的。还要注意，灰色存储单元的字节被确认后，这些存储单元可以回收并且对发送进程可用，这就是我们给出一个环形缓冲区的原因。

接收端的缓冲区操作比较简单。环形缓冲区分成两个区域（表示为白色和灰色）。白色区域包含空存储单元，可以由从网络上接收的字节进行填充。灰色区域表示接收到的字节，可以由接收进程读出。当某个字节被接收进程读出以后，这个存储单元可被回收，并加入到空存储单元池中。

段

尽管缓冲能够处理生产进程速度和消费进程速度之间的不相称问题，但在发送数据之前，还需要多个步骤。IP 层作为 TCP 服务的提供者，需要以分组的方式而不是字节流的方式发送数据。在传输层，TCP 将多个字节组合在一起成为一个分组，这个分组称为段（segment）。TCP 给每个段添加头部（为了达到控制目的），并将该段传递给 IP 层。段被封装到 IP 数据报中，然后再进行传输。整个操作对接收进程是透明的。稍后，我们会看到这些段可能被无序接收、丢失，或者损坏和重发。所有这些均由 TCP 处理，接收进程不会察觉到任何操作。图 3-43 表示了在缓冲区中如何从字节生成段。

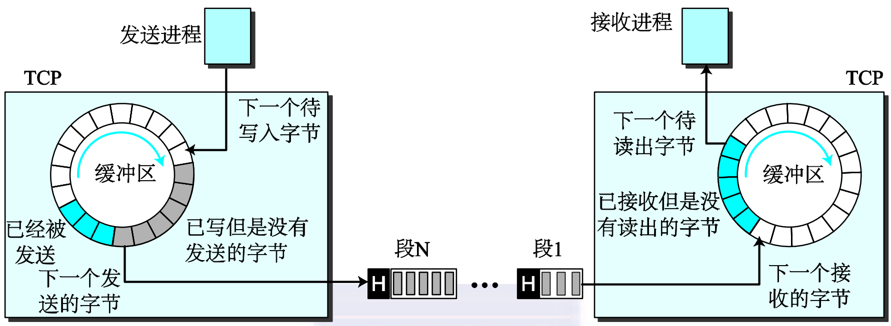


图 3-43　TCP 段

注意，段的大小不必相同。为了简单起见，我们在图中只表示了一个包含 3 个字节的段和另一个包含 5 个字节的段。实际的段可能包含数百（或者数千）个字节。

全双工通信

TCP 提供全双工服务（full-dupler service），即数据可以在同一时间双向流动。每一方向 TCP 都有发送和接收缓冲区，它们能在双向发送和接收段。

多路复用和多路分解

与 UDP 类似，TCP 在发送端执行多路复用，在接收端执行多路分解。然而，由于 TCP 是一个面向连接协议，因此需要为每对进程建立连接。

面向连接的服务

与 UDP 不同，TCP 是一种面向连接的协议。位于站点 A 的一个进程与站点 B 的另外一个进程想要进行数据的发送和接收，步骤如下：

1. 在两个 TCP 之间建立一个连接。

2. 在两个方向交换数据。

3. 连接终止。

注意，这是一个逻辑连接，而不是一个物理连接。TCP 段封装成 IP 数据段，并且可能被无序地发送，或丢失，或被破坏，然后重发。每个段都可以通过不同的路径到达目的端。TCP 建立一种面向字节流的环境，在这种环境中，TCP 能承担按顺序传递这些字节到其他站点的任务。

可靠的服务

TCP 是一种可靠的传输协议。它使用确认机制来检查数据是否安全和完整地到达。我们将在本章的后面进一步讨论差错控制的特点。

3.4.2　TCP 的特点

为了提供上一节所提及的服务，TCP 需要一些特性，本节将针对这些特性进行简要概述并在稍后进行详细讨论。

序号系统

虽然 TCP 软件能够记录发送或接收的段，但是在段的头部没有段序号字段。TCP 在段的头部采用称为序号（sequence number）和确认号（acknowledgment number）的两个字段。这两个字段指的是字节序号，而不是段序号。

字节序号

TCP 为在一个连接中传输的所有数据字节（八位字节）编号。在每个方向上序号都是独立的。当 TCP 接收来自进程的一些数据字节时，TCP 将它们存储在发送缓冲区中并给它们编号。不必从 0 开始编码，TCP 在 0 到 $2^{32}-1$ 之间生成一个随机数作为第一个字节的序号，例如，如果随机数是 1057，并且发送的全部字节个数是 6000，那么这些字节序号是 1057~7056。下面将会看到字节序号是用于流量和差错控制。

> 在每个连接中传送的字节都由 TCP 编号，序号开始于一个随机产生的数。

序号

字节被编号后，TCP 对发送的每一个段分配一个序号。在每一个方向上的序号定义如下：

1．第一段的序号是初始序号（initial sequence number，ISN），这是一个随机数。

2．其他段的序号是之前段的序号加之前段携带的字节数（实际上的或想象的）。之后，我们将给出一些控制段，它们被认为携带了一个想象字节。

例 3.17　假设一个 TCP 连接正在传送一个 5 000 字节的文件，第一个字节序号是 10 001。如果数据被分成 5 个段，每一个数据段携带 1 000 字节，试问每个段的序号是什么？

解答

每个段的序号如下所示：

段 1 → 序号：10 001	**范围：**	10 001	到	11 000
段 2 → 序号：11 001	**范围：**	11 001	到	12 000
段 3 → 序号：12 001	**范围：**	12 001	到	13 000
段 4 → 序号：13 001	**范围：**	13 001	到	14 000
段 5 → 序号：14 001	**范围：**	14 001	到	15 000

> 一个段的序号字段的值定义了该段包含的第一个字节的序号。

当一个段携带数据和控制信息（捎带）时，它使用一个序号。如果一个段没有携带用户数据，那么它逻辑上不定义序号。虽然字段存在，但是值是无效的。然而，当有些段仅携带控制信息时也需要有一个序号用于接收方的确认。这些段用作连接建立、连接终止或连接废弃。这些段中的每一个好像携带一个字节那样使用一个序号，但都没有实际的数据。当我们讨论连接的时候，我们将自己研究这个问题。

确认号

正如我们前面所讨论过的那样，TCP 中的通信是全双工的；当建立一个连接时，双方同时都能发送和接收数据。每一方为字节编号，每一方经常使用不同的起始字节号。每一方向的序号表明了该段所携带的第一个字节的序号。每一方也使用确认号来确认它已收到的字节。但是，确认号定义了该方预期接收的下一个字节的序号。另外，确认号是累积的，这意味着接收方记下它已安全而且完整地接收到最后一个字节的序号，然后将它加 1，并将这个结果作为确认号进行通告。在这里，术语“累积”指的是，如果一方使用 5 643 作为确认号，则表示它已经接收了所有从开始到序号为 5 642 的字节。但要注意，这并不是指接收方已经接收了 5 642 个字节，因为第一个字节的编号通常并不是从 0 开始的。

> 段中确认字段的值定义了通信一方预期接收的下一个字节的编号。确认号是累积的。

3.4.3　段

在我们更详细讨论 TCP 之前，让我们讨论 TCP 分组本身。在 TCP 中的分组称为**段**（segment）。

格式

段的格式如图 3-44 所示。段包含 20 字节到 60 字节的头部，接着是来自应用程序的数据。如果没有选项，那么头部是 20 字节；如果有选项，最多是 60 字节。在本节中，我们将讨论某些头部字段，通过本章的学习可以更加清楚这些字段的意义和目的。

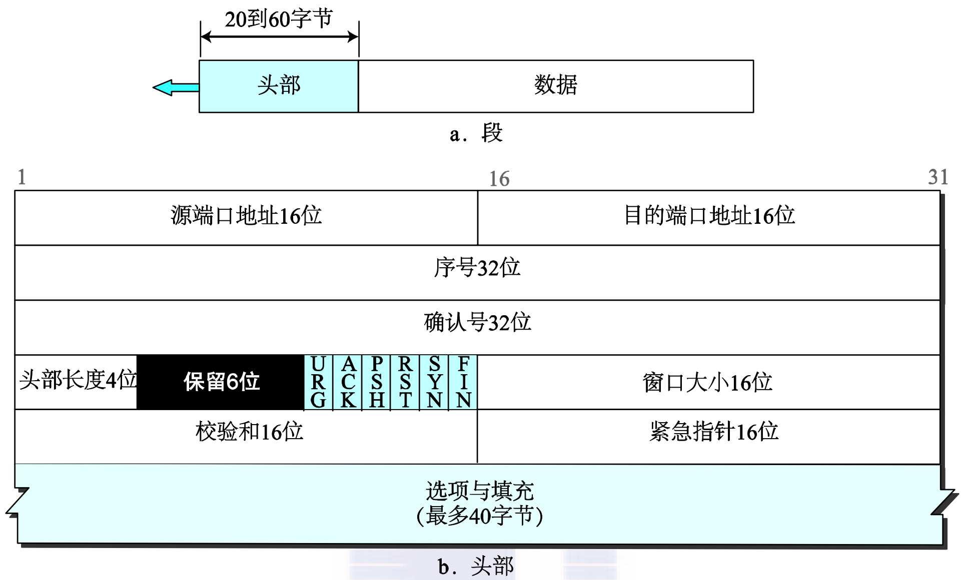


图 3-44　TCP 段格式

- **源端口地址**。这是一个 16 位的字段，它定义了在主机中发送该段的应用程序的端口号。这与 UDP 头部的源端口地址的作用一样。
- **目的端口地址**。这是一个 16 位的字段，它定义了在主机中接收该段的应用程序的端口号。这与 UDP 头部的目的端口地址的作用一样。
- **序号**。这个 32 位的字段定义了一个数，它分配给段中数据的第一个字节。如前所述，TCP 是一种字节流传输协议。为了确保连通性，对要发送的每一个字节都进行编号。序号告诉目的端，在这个序列中哪一个字节是该段的第一个字节。在连接建立时，每一方都使用随机数生成器产生一个**初始序号**（initial sequence number，ISN），通常每一个方向的 ISN 都不同。
- **确认号**。这个 32 位的字段定义了段的接收方期望从对方接收的字节号。如果段的接收方成功地接收了对方发来的字节号 x，它就将确认号定义为 $x+1$，确认和数据可捎带一起发送。
- **头部长度**。这个 4 位的字段指明了 TCP 头部中共有多少个 4 字节长的字。头部的长度可以在 20 字节到 60 字节之间。因此，这个字段的值在 5（$5\times4=20$）到 15（$15\times4=60$）之间。
- **控制**。这个字段定义了 6 种不同的控制位或标记，如图 3-45 所示。在同一时间可以设置一位或多位。这些位用在 TCP 的流量控制、连接建立和终止、连接失败和数据传送方式等方面。图 3-45 给出了每个位的说明。本章中，当讨论 TCP 的具体操作时，我们将对它们做深入的讨论。

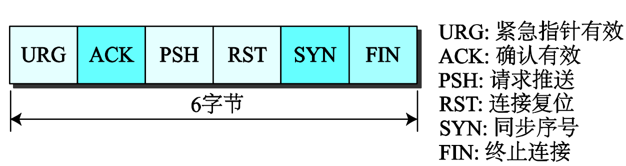


图 3-45　控制字段

- **窗口大小**。这个字段定义对方必须维持的窗口的大小（以字节为单位）。注意，这个字段的长度是 16 位，这意味着窗口的最大长度是 65 535 字节。这个值通常称为接收窗口（rwnd），它由接收方确定。此时，发送方必须服从接收端的支配。
- **校验和**。这个 16 位的字段包含了校验和。TCP 校验和的计算过程与前面描述的 UDP 所采用的计算过程相同。但是，在 UDP 数据报中校验和是可选的。然而，对 TCP 来说，将校验和包含进去是强制的。起相同作用的伪头部被加到段上。对 TCP 伪头部，协议字段的值是 6。如图 3-46 所示。

在 TCP 中使用校验和是强制的。

- **紧急指示符**。这个 16 位的字段只有当紧急标志置位时才有效，这个段包含了紧急数据。它定义了一个数，将此数加到序号上就得出此段数据部分中最后一个紧急字节，在本章稍后将会讨论它。
- **选项**。在 TCP 头部中可以有多达 40 个字节的可选信息。我们将在本节稍后讨论 TCP 头部中使用的某些选项。

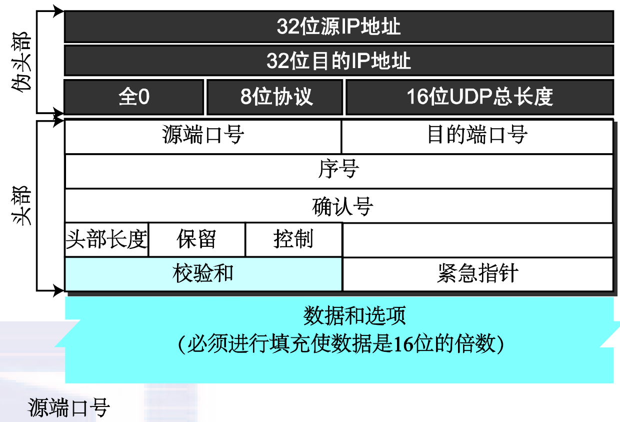


图 3-46　加到 TCP 数据报上的伪头部

封装

一个 TCP 段封装了来自应用层的数据。TCP 段被封装在 IP 数据报中，IP 数据报被封装在数据链路层的帧中。

3.4.4　TCP 连接

TCP 是一种面向连接的协议。面向连接的传输协议在源端和目的端之间建立一条虚路径。然后，属于一个报文的所有段都沿着这条虚路径发送。整个报文使用单一的虚路径有利于确认处理以及对损坏或丢失帧的重发。读者可能想知道 TCP 如何使用 IP 服务，一个无连接协议如何能面向连接。关键就在于 TCP 的连接是虚连接，不是物理连接。TCP 在一个较高层次上操作，TCP 使用 IP 服务向接收方传递独立的段，但它控制连接本身。如果一个段丢失了或损坏了，则重新发送它。与 TCP 不同，IP 不知道这个重新发送过程。如果一个段失序到达，则 TCP 保存它直到缺少的段到达。IP 是不知道这个重新排序过程的。

在 TCP 中，面向连接的传输需要三个过程：连接建立、数据传输和连接终止。

连接建立

TCP 以全双工方式传输数据。当两个机器中的两个 TCP 建立连接后，它们就能够同时向对方发送段。这就表示，在传输数据之前，每一方都必须对通信进行初始化，并得到对方的认可。

三次握手

在 TCP 中的连接建立称为**三次握手**（three-way handshaking）。在我们的例子中，称为客户的应用程序想要与另一个称为服务器的应用程序使用 TCP 作为传输层协议建立连接。

该过程从服务器开始。服务器程序告诉它的 TCP，它已准备好接收一个连接。这就称为*被动打开*（passive open）。虽然 TCP 已经准备好接收从世界上任何一个机器发来的连接，但它自己并不能完成这个连接。

客户程序发出请求进行*主动打开*（active open）。想要与服务器进行连接的客户告诉它的 TCP，

它需要连接到特定的服务器。TCP 现在就开始如图 3-47 所示的三次握手过程。

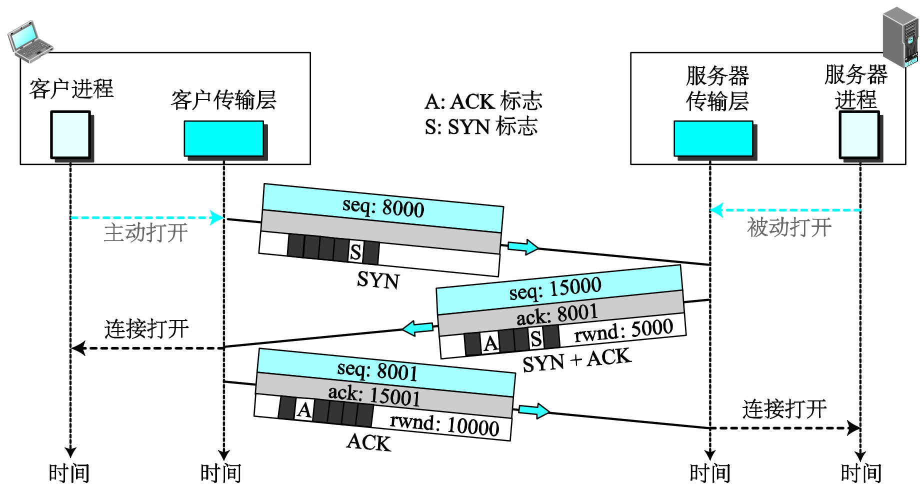


图 3-47　使用三次握手建立连接

为了表示该过程，我们使用了两个时间序列，每端一个。每个段头部的所有字段都有值，或许它的某些选项字段也有。但是，我们每次仅表示少数几个必须要知道的字段，如果序号、确认号、控制标记（仅仅是其中被置位的）和窗口大小等有值，那么我们就表示它们。这个阶段的三个步骤如下。

1．客户发送的第一个段是 SYN 段。这个段仅有 SYN 标志被置位，它用于序号同步。它占用一个序号。当数据传输开始时，在我们的例子中，客户随机选择一个数字作为初始序号（ISN）。注意，这个段不包含确认号。它也没有定义窗口大小；窗口大小的定义只有当段包含确认号时才有意义。段也能包含一些我们本章稍后讨论的选项。注意，SYN 段是一个控制段并且不携带数据。然而，它消耗一个序号，因为它需要被确认。我们可以说 SYN 段携带了一个假想字节。

SYN 段不携带数据，但它占用一个序号。

2．服务器发送第二个段，两个标志位 SYN 和 ACK 置位的段，即 SYN +ACK 段。这个段有两个目的。首先，它是另一方向通信的 SYN 段。服务器使用这个段来初始化序号，这个序号用来给从服务器发向客户的字节编号。服务器也通过给 ACK 置位并展示下一个序号来确认接收到来自客户的 SYN 段，这里的下一个序号是服务器预期从客户接收的序号。我们将在介绍流量控制那一节看到，因为它包含确认，它也需要定义接收窗口，即 rwnd（客户使用）。因为这个段起到 SYN 段的作用，它需要被确认。因此，它占用一个序号。

SYN + ACK 段不携带数据，但它占用一个序号。

3．客户发送第三个段。这个段仅仅是一个 ACK 段。它使用 ACK 标志和确认序号字段来确认收到了第二个段。注意，如果不携带数据，ACK 段没有占用任何序号，但是一些实现允许这第三个段在连接阶段从客户端携带第一块数据。在这种情况下，段消耗的序号与数据字节数相同。

ACK 段，如果不携带数据，则它不占用序号。

SYN 泛洪攻击

在 TCP 中，连接建立过程易遭受到称为 **SYN 泛洪攻击**（SYN flooding attack）的严重安全问题。一个恶意的攻击者将大量的 SYN 段发送到一个服务器，在数据报中通过伪装源 IP 地址假装这些 SYN 段是来自不同的客户端，此时就是 SYN 泛洪攻击。假定客户机发出主动打开，服务器分配

必要的资源，如生成转换控制块（TCB）和设置计时器等。然后服务器发送 SYN+ACK 段给这些假客户，但这些段都丢失了。然而，当服务器等待第三段握手过程时，许多资源被占用但没有被使用。如果在短时间内，SYN 段的数量很大，服务器最终会耗尽资源而崩溃。这种 SYN 泛洪攻击属于一种称为**拒绝服务攻击**（denial of service attack）的安全攻击类型，其中，一个攻击者独占系统如此多的服务请求使得系统崩溃，拒绝对每个请求提供服务。

TCP 的某些实现拥有减轻 SYN 攻击影响的策略。有些实现在特定时间周期内对连接请求进行限制，另一些实现过滤掉来自不需要的源地址的数据报。一个新的策略使用 cookie 推迟资源分配直到一个完整的连接建立。新的传输层协议 SCTP 使用这种策略，我们将在第 8 章讨论。

数据传输

连接建立后，可进行双向数据传输，客户端与服务器双方都可发送数据和确认。在本章稍后，我们将学习确认的规则。目前，知道在同一段内携带确认时，在同一方向上也和可以传递数据就够了。这就是数据捎带确认。图 3-48 给出了一个例子。

在这个例子中，在连接建立后，客户端用两个段发送 2000 个字节的数据。然后，服务器用一个段发送 2 000 个字节的数据。客户端发送另一个段。前面三个段携带数据与确认，但是最后一个段仅携带确认，这是因为已没有数据发送了。注意序号与确认号数值，客户端发送的数据段有 PSH（推送）标志，所以服务器 TCP 知道在接收到数据时立刻传递给服务器进程。稍后我们讨论这个标志的用法。另一方面，来自服务器的段没有设置推送标志。大多数 TCP 的实现都有可选标志，可设置或不设置。

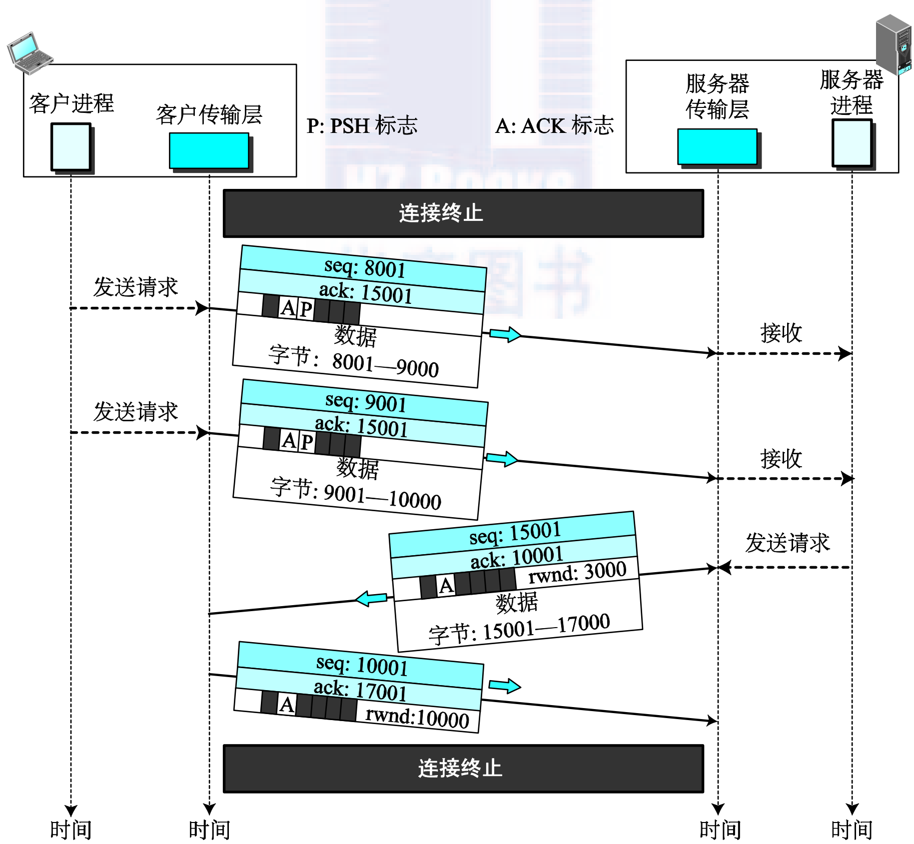


图 3-48　数据传输

推送数据

我们看到发送方的 TCP 使用缓冲区存储来自发送方应用程序的数据流。发送方的 TCP 可以选择段的大小。接收方的 TCP 在数据到达时也将数据进行缓存，并当应用程序准备就绪时或当接收端 TCP 认为方便时将这些数据传递给应用程序。这种灵活性增加了 TCP 的效率。

但是，在有些情况下，应用程序并不需要这种灵活性。例如，应用程序与另一方应用程序进行交互式通信。一方的应用程序打算将其击键发给对方应用程序，并希望接收到立即响应。数据的延迟传输和延迟传递对这个应用程序来说是不可接受的。

TCP 可以处理这种情况。在发送端的应用程序可请求推送操作。这就表示发送端的 TCP 不必等待窗口被填满。它创建一个段就立即将其发送。发送端的 TCP 还必须设置推送位（PSH）以告诉接收端的 TCP，这个段所包含的数据必须尽快地传递给接收应用程序，而不要等待更多数据的到来。这意味着将面向字节的 TCP 改为面向块的 TCP，但是 TCP 可以选择使用或不使用这个特性。

紧急数据

TCP 是面向字节流的协议。这就是说，从应用程序到 TCP 的数据被表示成一串字节流。数据的每一个字节在流中占有一个位置。但是，在某些情况下，应用程序需要发送紧急（urgent）字节，某些字节需要另一端的应用以特殊方式对待。解决方法是发送一个 URG 标志置位的段。发送应用程序告诉发送端的 TCP，这块数据是紧急的。发送端 TCP 创建段，并将紧急数据放在段的开始。段的其余部分可以包括来自缓冲区的普通数据。头部中的紧急指针字段定义了紧急数据的结束（紧急数据的最后一个字节）。例如，如果段序号是 15 000 且紧急指针的值是 200，那么紧急数据的第一字节是字节 15 000 且最后一个字节是 15 200。段中的剩余字节（如果存在的话）是非紧急的。

有一点需要提及，这很重要，那就是 TCP 的紧急数据不是人们所想的优先服务，也不是带外数据服务。相反，TCP 的紧急模式是一种服务，发送端的应用程序通过这种服务将字节流的某些部分标记为需要接收端特别对待的字节流。接收端 TCP 将字节（紧急或非紧急）按序传递到应用程序，但是通知应用程序紧急数据的开始和结束。留给应用程序决定如何处理紧急数据。

连接终止

交换数据双方的任一方（客户或服务器）都可关闭连接，尽管通常情况下是由客户端发起。当前大多数对连接终止的实现有两个方法：三次握手和带有半关闭选项的四次握手。

三次握手

当前对连接终止的绝大多数实现是三次握手（three-way handshaking），如图 3-49 所示。

1．在正常情况下，在客户进程接收到一个关闭命令后，客户的 TCP 发送第一个段：FIN 段，即其中的 FIN 标志置位。注意，FIN 段可包含客户机要发送的最后数据块，或如图 3-49 所示的只是控制段。如果它只是控制段，它仅占有一个序号因为它需要被确认。

> 如果 FIN 段不携带数据，则该段占用一个序号。

2．服务器 TCP 接收到 FIN 段后，通知它的进程，并发送第二个段：FIN + ACK 段，证实它接收到来自客户端的 FIN 段，同时通告另一端连接关闭。这个段还可以包含来自服务器的最后数据块。如果它不携带数据，则这个段仅占用一个序号。

> 如果 FIN + ACK 段没有携带数据，则该段仅占用一个序号。

3．客户端的 TCP 发送最后一段，即 ACK 段，来证实它接收到来自服务器的 FIN 段。这个段包含确认号，它是来自服务器的 FIN 段的序号加 1。这个段不携带数据也不占用序号。

半关闭

在 TCP 中，一端可以停止发送数据后，还可以接续接收数据。这就是所谓的**半关闭**（half-close）。

虽然任一端都可发出半关闭，但通常都是由客户端发起的。当服务器在开始处理之前需要接收到所有数据，这时就会出现半关闭。例如，排序是一个很好的例子。客户端发送数据给服务器进行排序，在开始排序之前，服务器需要接收到全部数据。这就是说，客户端发送全部数据之后，它在客户到服务器方向可关闭连接。但为了返回存储数据，服务器到客户方向必须保持打开。服务器在接收到数据后还需要时间进行排序；它的向外方向必须保持打开。图 3-50 给出了半关闭的例子。

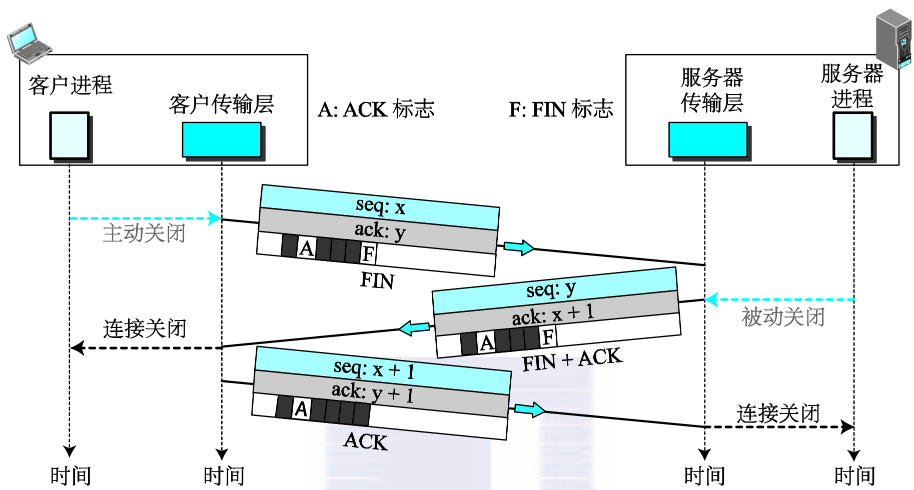


图 3-49　使用三次握手的连接终止

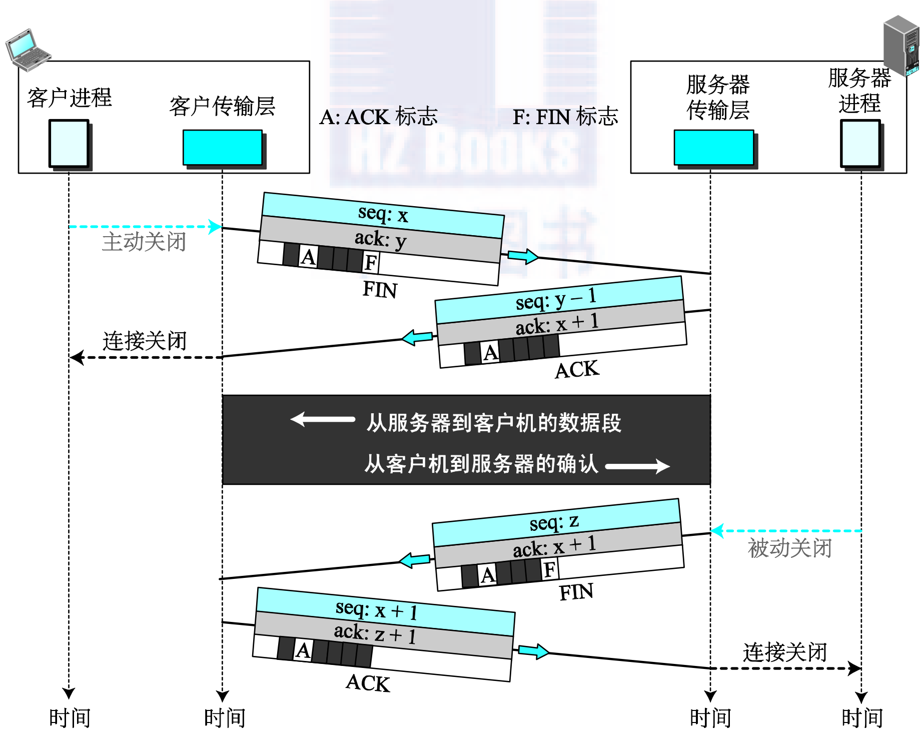


图 3-50　半关闭

从客户到服务器的数据传输停止。客户端通过发送 FIN 段实现半关闭连接。服务器通过发送 ACK 段确认半关闭。然而，服务器还可以发送数据。当服务器已经发送完被处理的数据时，它发

送一个 FIN 段。该 FIN 段由客户端的 ACK 来确认。

连接半关闭后，数据可以从服务器传送给客户端，而确认可以从客户端传送给服务器。客户不能再向服务器发送任何数据。

连接重置

在一端的 TCP 可能拒绝连接请求，可能终止已存在的连接，也可能结束空闲连接。所有这些都通过 RST（重置）标志完成。

3.4.5　状态转换图

为了记录发生在连接建立、连接终止和数据传输阶段发生的事件，如图 3-51 所示，TCP 以有限状态机的形式进行详述。

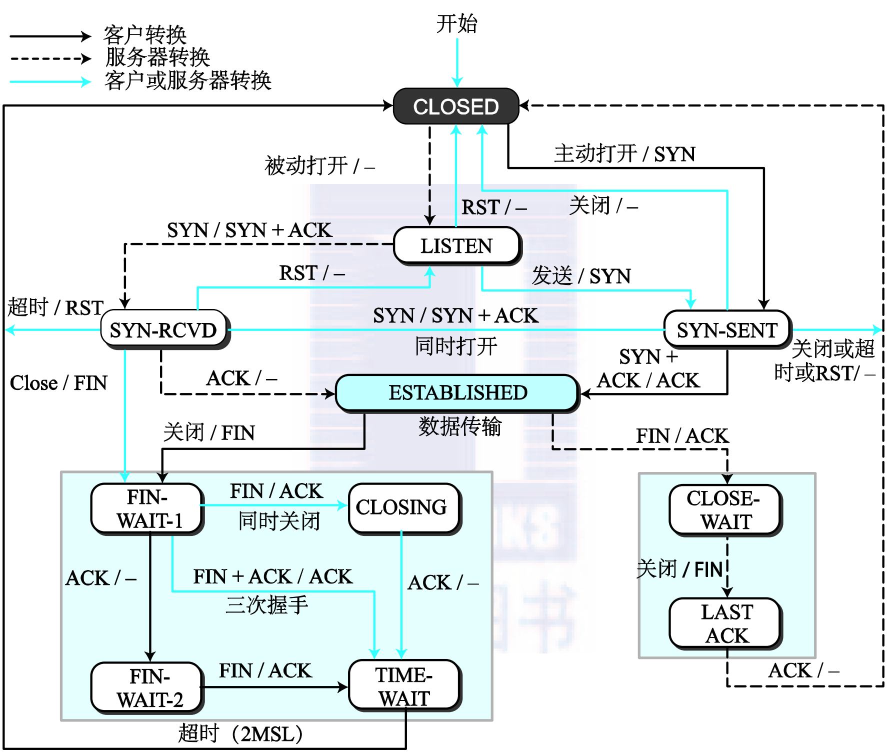


图 3-51　状态转换图

图 3-51 给出了 TCP 客户和服务器使用的两个有限状态机，TCP 客户和服务器被结合到一幅图中。圆角矩形代表状态。从一个状态到另一个状态的转换使用直线表示。每条线有两个字符串，它们用斜线分隔。第一个字符串是输入，即 TCP 所接收的内容。第二个字符串是输出，即 TCP 所发送的内容。图中虚线代表服务器通常经历的转换；黑色实线给出客户通常经历的转换。然而，在某种情况下，服务器沿着实线进行状态转换，客户沿着虚线进行状态转换。灰色线给出特殊情况。注意，被标记为 ESTABLISHED 的圆角矩形实际上是两种状态，一个是客户状态，另一个是服务器状态，它们用于流量和差错控制，本章之后会对其进行解释。我们将会在本章末尾讨论图中提及的计时器，包含 2MSL（2 Maximum Segment Lifetime，2 倍段最大生存时间）计时器。我们使用基于图 3-51 的多个场景并在每个情况中展示图 3-51 的一部分。

在有限状态机中被标记为 ESTABLISHED 状态实际上是两个不同状态，这些状态是客户和服务器进行传输数据经历的。

表 3-2 给出的 TCP 的状态表。

场景

为了理解 TCP 状态机以及转换图，我们在本节考察一种场景。

表 3-2　TCP 状态

状　　态	说　　明	状　　态	说　　明
CLOSED	没有连接存在	**FIN-WAIT-2**	首个 FIN 的 ACK 已被接收；等待第二个 FIN
LISTEN	接收到被动打开；等待 SYN	**CLOSE-WAIT**	首个 FIN 被接收，ACK 被发送；等待应用关闭
SYN-SENT	SYN 已被发送；等待 ACK	**TIME-WAIT**	第二个 FIN 被接收，ACK 被发送；等待 2MSL 超时
SYN-RCVD	SYN + ACK 已被发送；等待 ACK	**LAST-ACK**	第二个 FIN 被发送；等待 ACK
ESTABLISHED	连接建立；数据传输正在进行	**CLOSING**	双端决定同时关闭
FIN-WAIT-1	首个 FIN 已被发送；等待 ACK		

一个半关闭场景

图 3-52 给出了这个场景的状态转换图。

客户进程向它的 TCP 发出*主动打开*命令来请求连接到特定套接字地址。TCP 发送一个 SYN 段并转移到 **SYN-SENT** 状态。在收到 SYN + ACK 段后，TCP 发送了一个 ACK 段并且进入 **ESTABLISHED** 状态。数据被传输，可能是双向的，并且被确认。当客户进程没有数据要发送了，它发出称为*主动关闭*的命令。TCP 发送 FIN 段并进入 **FIN-WAIT-1** 状态。当它接收到 ACK 段，它进入 **FIN-WAIT-2** 状态。当客户接收到 FIN 段时，它发送一个 ACK 段并进入 **TIME-WAIT** 状态。客户保持这种状态 2MSL 秒（见本章结尾的 TCP 计时器）。当相应计时器超时，客户进入 **CLOSED** 状态。

服务器进程发出*被动打开*命令。服务器 TCP 进入 **LISTEN** 状态并且保持这种状态直到它接收到一个 SYN 段。TCP 之后发送一个 SYN + ACK 段并且进入 **SYN-RCVD** 状态，等待客户发送 ACK 段。在接收到 ACK 段后，TCP 进入 **ESTABLISHED** 状态，这就开始了数据传输。TCP 保持这种状态直到它接收到一个来自客户的 FIN 段，这表示没有其他数据要被交换且连接可以被关闭。一旦服务器接收到 FIN 段，那么它就向客户发送带有虚拟 EOF 标记的排队中所有的数据，这意味连接必须被关闭。它发送一个 ACK 段且进入 **CLOSE-WAIT** 状态，但是推迟确认来自客户的 FIN 段，直到它接收到来自进程的*被动关闭*命令。在接收到被动关闭命令后，服务器向客户发送 FIN 段并进入 **LAST-ACK** 状态，等待最终 ACK。当 ACK 段被从客户接收，服务器进入 **CLOSE** 状态。图 3-53 使用时间线上的状态给出相同的场景。

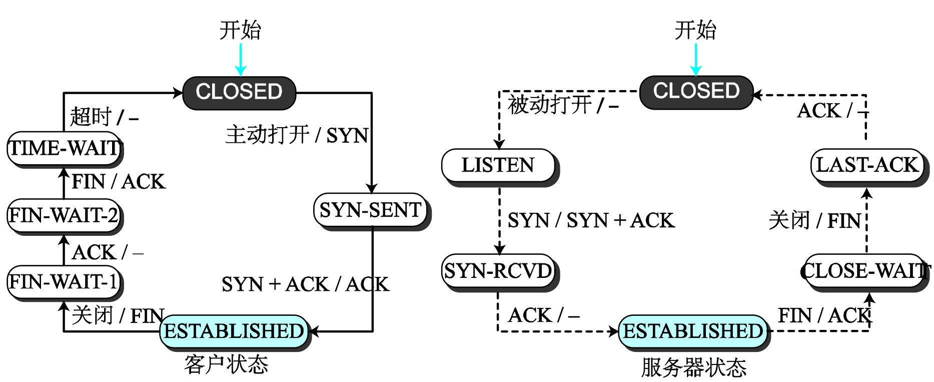


图 3-52　半关闭连接终止转换图

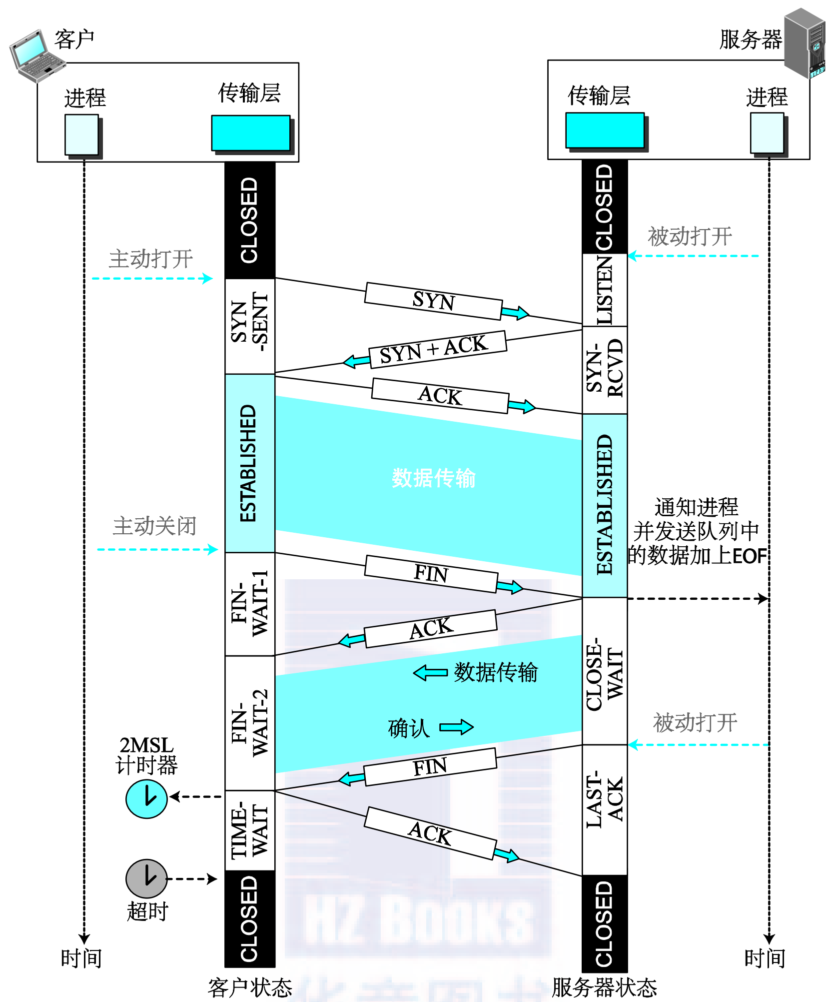


图 3-53　时间线图表示的普通场景

3.4.6　TCP 中的窗口

在讨论 TCP 中的数据传输并提出诸如流量、差错和拥塞控制问题之前，我们首先描述 TCP 中使用到的窗口。TCP 在每个方向的数据传输上使用两个窗口（发送窗口和接收窗口），这意味着双向通信有四个窗口。为了使得讨论简单，我们做一个不实际的假设，即通信只是单向的（比如从客户到服务器）；双向通信可以使用两个带有捎带的单向通信推理出来。

发送窗口

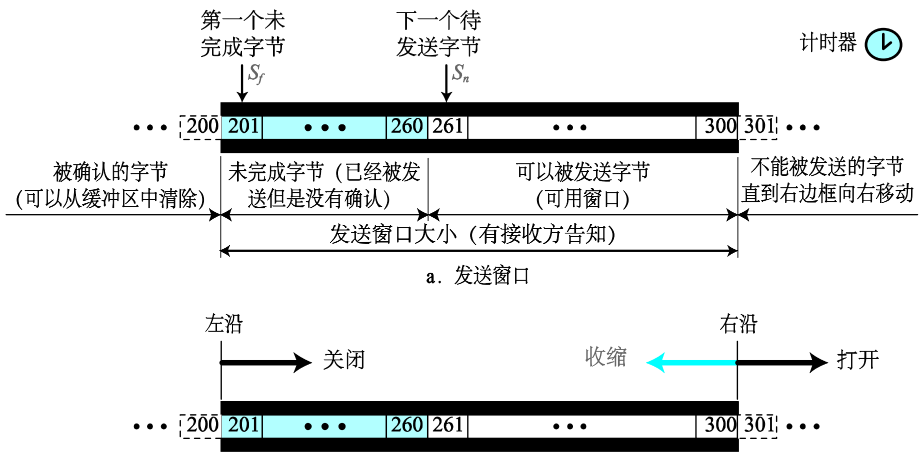
图 3-54 给出了一个发送窗口的例子。窗口大小是 100 字节，但是之后我们会看到发送窗口大小由接收方（流量控制）和底层网络的拥塞程度（拥塞控制）指定。图 3-54 给出了发送窗口如何打开、关闭以及收缩。

TCP 中的发送窗口与选择性重复协议中的发送窗口相似，但是有一些不同：

1．一个区别是与窗口相关的实体本质不同。在 SR 中窗口的大小是分组的数量，但是 TCP 中的窗口大小是字节的数量。尽管 TCP 中实际传输是一段接一段发生的，但是控制窗口的变量是以字节为单位的。

2．第二个区别是，在某些实现中，TCP 可以存储来自进程的数据并且在之后发送它们，但是我们假设发送方 TCP 一旦从进程中接收到数据就能够发送数据段。

3．另一个区别是计时器的数量。理论上，选择性重复协议可能为每个被发送的分组使用多个计时器，而 TCP 只使用一个计时器。



a．发送窗口

b．打开、关闭并且收缩发送窗口

图 3-54　TCP 中的发送窗口

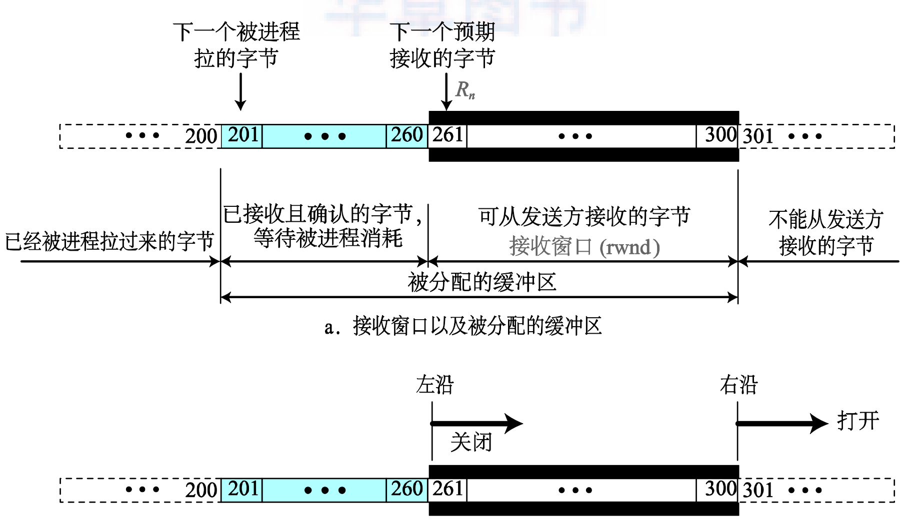
接收窗口

图 3-55 给出了一个接收窗口的例子。窗口大小是 100 字节。图 3-55 也给出了接收窗口如何打开以及关闭；实际上，窗口从不收缩。

TCP 中的接收窗口与 SR 中的接收窗口有两点不同。

1．第一个区别是 TCP 允许接收进程以自己的速率拉数据。这意味着接收方部分被分配缓冲区可以被已接收且确认的字节占据，但是它们正在等待被接收进程拉过去。如图 3-55 所示，接收窗口大小总是小于或等于缓冲区大小。接收窗口大小决定了接收窗口在被淹没（流量控制）之前可以从发送方接收的字节数量。换言之，接收窗口通常称为 rwnd，可以由下式决定：

$$\text{rwnd} = \text{缓冲区大小} - \text{等待被拉字节数量}$$



a．接收窗口以及被分配的缓冲区

b．打开并关闭接收窗口

图 3-55　TCP 中的接收窗口

2．第二个不同是在 TCP 协议中使用的确认方式不同。请记住，在 SR 中的确认是选择性的，它定义了已经被接收的分组。TCP 中主要确认机制是累积确认，它声明了下一个预期接收字节（我们之前讨论过，按这种方式 TCP 看起来像 GBN）。然而，TCP 的新版本使用了累积确认和选择性确认；我们在本书的网站上讨论了这些选项。

3.4.7　流量控制

如之前所讨论的，流量控制平衡了生产者创建数据的速率与消费者使用数据的速率。TCP 将流量控制与差错控制分开。在本节，我们讨论流量控制，忽略差错控制。我们假设发送和接收 TCP 之间的逻辑信道是无错的。

图 3-56 给出了发送方和接收方之间按的单向数据传输；双向数据传输可以从这个单向进程中推断出来。

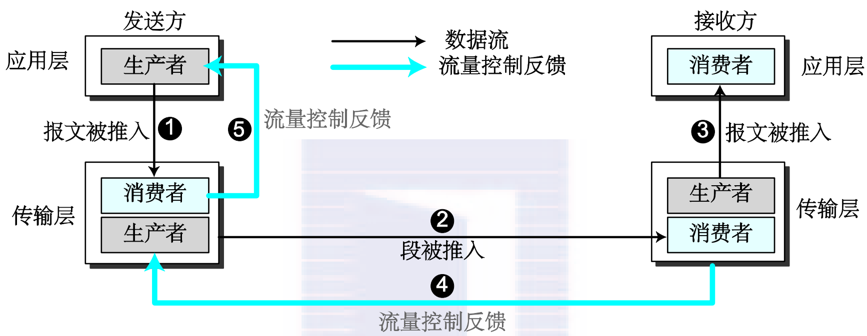


图 3-56　TCP 中数据流和流量控制反馈

图 3-56 给出了从发送方进程到发送方 TCP、从发送方 TCP 到接收方 TCP 以及从接收方 TCP 上升到接收方进程的数据（路径 1、2 和 3）传输过程。然而，流量控制反馈却是从接收方 TCP 传输到发送方 TCP 并且从发送方 TCP 上升到发送方进程（路径 4 和 5）。绝大多数 TCP 实现不提供从接收方进程到接收方 TCP 的流量控制反馈；无论何时，当接收方进程准备好了，具体实现就会让接收方进程从接收方 TCP 中拉数据。换言之，接收方 TCP 控制发送方 TCP；发送方 TCP 控制发送方进程。

从发送方 TCP 到发送方进程（路径 5）的流量控制反馈的实现方式是，当窗口已满，它简单拒绝发送方 TCP 的数据。这意味着，我们对于流量控制的讨论集中于由接收方 TCP 发向发送方 TCP 的反馈。

打开以及关闭窗口

为了实现流量控制，TCP 迫使发送方和接收方调整它们的窗口大小，尽管当连接建立时两方的缓冲区大小是固定的。当更多的数据从发送方到来时，接收方窗口关闭（向右移动左沿）；当更多的数据被进程拉过来时，它打开窗口（向右移动右沿）。我们假设它不会收缩（右沿不会向左移动）。

发送窗口的打开、关闭和收缩由接收方控制。当一个新的确认允许发送窗口关闭时，发送窗口关闭（向右移动左沿）。当接收方通知的接收窗口大小（rwnd）允许发送方窗口打开时（新 ackNo + 新 rwnd > 上一个 ackNo + 上一个 rwnd），发送窗口打开（向右移动右沿）。这种情况没有发生的事件中，发送窗口缩小。

一种场景

我们给出发送方和接收方窗口如何在连接建立阶段设置大小，并给出它们在数据传输阶段变化的情况。图 3-57 给出了一个简单的单向数据传输例子（从客户到服务器）。我们暂时忽略差错控制，假设没有段被破坏、丢失、重复或失序到达。注意，我们给出两个单向数据传输的窗口。尽管客户

在第三个段将服务器窗口设为 2000，但是我们不给出那个窗口，因为通信是单向的。

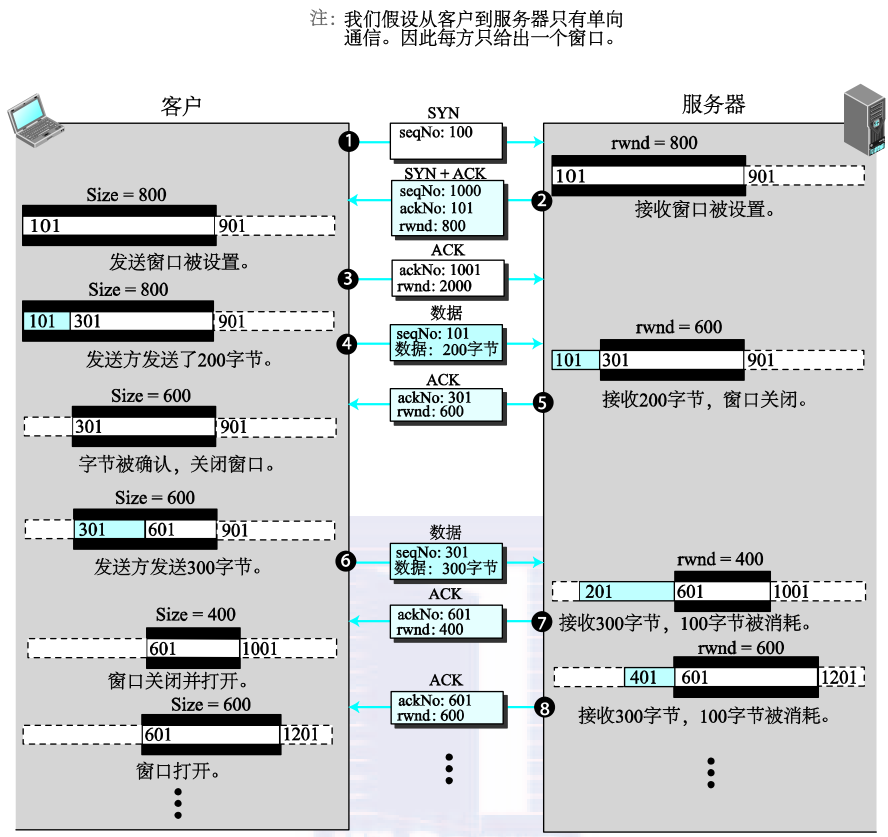


图 3-57　流量控制举例

客户和服务器之间交换了 8 个段。

1．从客户到服务器的第 1 段（SYN 段）请求连接。这个段声明起始 seqNo ＝ 100。当这个段到达服务器时，它分配了大小为 800 字节的缓冲区（假设）并设置窗口覆盖了全部缓冲区（rwnd = 800）。注意，下一个要到达的字节是 101。

2．第 2 段是从服务器到客户的。这是 ACK + SYN 段。段使用 ackNo = 101，这表示它期待接收的字节从 101 开始。它也声明了客户可以设置大小为 800 字节的缓冲区。

3．第 3 段是从客户到服务器的 ACK 段。注意，客户可以定义大小为 2000 的 rwnd，但是，在图 3-57 中我们不使用这个值，因为通信是单向的。

4．在客户设置了服务器指定的窗口大小（800）之后，进程推送 200 字节数据。TCP 客户给这些字节编号为从 101 到 300。之后，它创建了一个段并发送到服务器。段开始字节数是 101 并且携带了 200 字节。之后客户窗口调整，表示 200 字节数据被发送但是等待确认。当这个段被服务器接收，这些字节被存储，并且接收窗口关闭来表示下一个预期字节是 301；存储字节占据了缓冲区的 200 字节。

5．第 5 段是从服务器到客户的反馈。服务器确认了多达 300 个字节，含第 300 个字节（预期接收 301 字节）。这个段也携带了减少后的接收窗口大小（600）。在接收这个段之后，客户从它的窗口中清除确认字节并关闭它的窗口，这表示下一个待发送字节是 301。然而，窗口大小减少到 600 字节。尽管分配缓冲区可以存储 800 字节，但是窗口不能打开（向右移动右沿），因为接收方不允许。

6. 在进程又推送了 300 个字节后，第 6 段被客户发送。段定义了 seqNo 为 301，并包含了 300 字节。当这个段到达服务器，服务器存储它们，但是它必须减少自己的窗口大小。在进程拉 100 字节数据后，窗口左边关闭了 300 字节，但是右侧打开了 100 字节。结果是窗口大小只减少了 200 字节。现在接收窗口大小是 400 字节。

7. 在第 7 段，服务器确认接收数据，并声明它的窗口大小是 400。当这个段到达客户，客户别无选择，只能再次减少它的窗口，并令窗口大小为服务器通告的 rwnd = 400。发送窗口从左侧关闭 300 字节，并从右侧打开 100 字节。

8. 在进程拉另外 200 字节后，第 8 段也是从来自服务器的。它的窗口大小增加。现在，新的 rwnd 的值是 600。段告知客户，服务器仍然期待 601 字节，但是服务器窗口大小增加到 600。我们需要提及的是，这个段的发送依赖于具体实现所规定的策略。一些实现可能不允许在这个时候通告 rwnd；服务器需要在这样做之前接收一些数据。在这个段到达客户之后，客户将窗口打开 200 字节而不关闭。结果是窗口增大到 600 字节。

窗口收缩

如前所述，接收窗口不能收缩。另一方面，如果接收方为 rwnd 定义了导致窗口收缩的数值，那么发送窗口可以收缩。然而，一些实现不允许收缩发送窗口。这个限制不允许发送窗口的右沿向左移动。换言之，接收方需要保持上一个和新的确认之间以及上一个和新 rwnd 值之间的如下关系，以防止发送窗口收缩。

$$\text{新 ackNo} + \text{新 rwnd} \geqslant \text{上一个 ackNo} + \text{上一个 rwnd}$$

不等式左侧表示与序号空间相关的右沿位置；右边给出右沿的旧位置。这个关系表示右沿不能向左移动。不等式是对接收端的命令，它使接收端检查自己的通告。然而，注意，只有当 $S_f < S_n$ 时不等式是有效的；我们需要记住所有计算都是模 2^{32}。

例 3.18 图 3-58 给出了这个命令的原因。

图 3-58 中 a 部分给出了上次确认和 rwnd 的值。b 部分给出了一种情况，其中发送方已经发送了 206 到 214 的字节。206 到 209 字节被确认并被清除。然而，新的通告定义 rwnd 的新值为 4，其中，210 + 4 < 206 + 12。当发送窗口收缩时，它产生了一个问题，已经被发送的 214 字节处于窗口之外。我们之前讨论的关系迫使接收方保持窗口右边沿与 a 部分相同，因为接收方不知道 210 到 217 之间哪些字节已经被发送。防止这种情况的一种方法是，让接收方推迟反馈，直到在它的窗口中有足够的缓冲区位置。换言之，接收方应该等待，直到更多的字节被它的进程消耗，从而来满足上面描述的关系。

窗口关闭

我们说过，不鼓励将右沿向左移动来收缩发送窗口。然而，有一个例外：接收方可以通过发送 rwnd 为 0 来临时关闭窗口。这只会在某些原因下发生，即接收方在一段时间内不想接收来自发送方的任何数据。在这种情况下，发送方并不真的收缩窗口大小，但是它停止发送数据直到新的通告到达。我们将在后面看到，即使当窗口因为来自接收方的命令而关闭了，发送方也总可以发送一个 1 字节数据的数据段。这称为探测，用来防止死锁（见 TCP 计时器部分）。

糊涂窗口综合征

当发送方应用程序缓慢创建数据时，或当接收方应用程序缓慢消耗数据时，或者两者同时发生时，在滑动窗口操作中可能发生一个严重的问题。任何这种情况都会导致以很小的段发送数据，这会降低操作的效率。例如，如果 TCP 发送一个只包含一字节数据的段，这意味着 41 字节数据报（20 字节 TCP 头部以及 20 字节 IP 头部）仅仅传输了 1 字节用户数据。此时，开销是 41/1，这表示我们非常低效地使用网络容量。在考虑到数据链路层和物理层开销后这种低效更为严重。这个问题称为**糊涂窗口综合征**（silly window syndrome）。对每一个站点，我们首先描述这个问题是如何产生的，

之后给出建议解决方法。

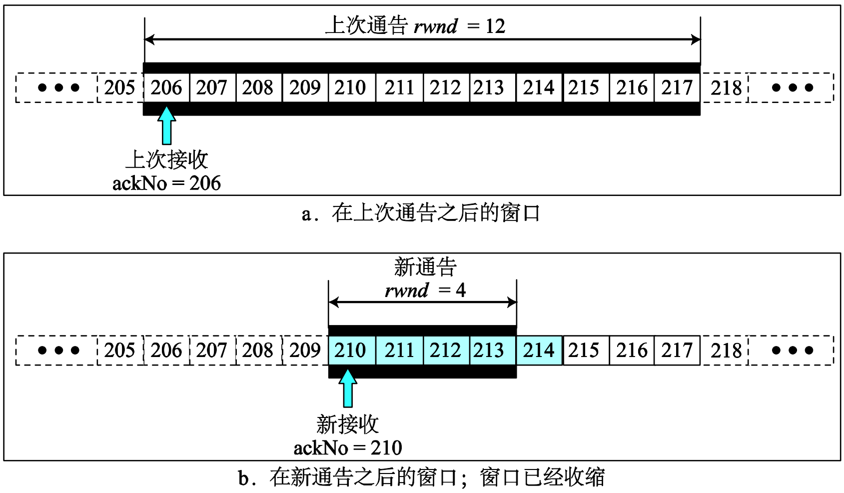


图 3-58　例 3.18

发送方产生的综合征

如果发送方 TCP 正在为一个创建数据很缓慢的应用程序服务，那么可能产生糊涂窗口综合征，例如，应用程序每次产生 1 字节数据。应用程序一次将 1 字节写入发送 TCP 的缓冲区。如果发送 TCP 没有任何特定指令，那么它可能创建包含 1 字节数据的段。结果是很多 41 字节的段在网络中传输。

解决方法是防止发送 TCP 一个字节一个字节地发送数据。发送 TCP 必须被迫等待并将数据收集成一个较大的块来发送。发送 TCP 要等待多长时间？如果等待太长，可能会延误进程。如果等待得不够长，可能结果就是发送小的段。Nagle 找到了一个巧妙的解决方法。**Nagle 算法**（Nagle's algorithm）非常简单：

1．即使从发送方应用程序接收来的第一个数据片段只有 1 字节，发送 TCP 也发送它们。

2．在发送第一个段之后，发送 TCP 在输出缓冲区积累数据并等待，直到接收 TCP 发送一个确认或直到积累了足够的数据来填充一个最大的段。此时，发送 TCP 可以发送段。

3．在剩余的传输中重复第二步。如果接收到对第 2 段的确认或者积累了足够的数据填充一个最大段，就立即发送第 3 段。

Nagle 算法的巧妙之处在于它的简单，并且事实上，它考虑了应用程序创建数据的速率以及网络传输数据的速率。如果应用程序比网络快，段就更大（最大段）。如果应用程序比网络慢，段就更小（小于最大段大小）。

接收方产生的综合征

接收方 TCP 可能产生糊涂窗口综合征，如果它正在为一个消耗数据很缓慢的应用程序服务，例如，应用程序每次消耗 1 字节数据。假设发送应用程序以 1 千字节的块来创建数据，但是接收应用程序每次只消耗 1 字节数据。也假设接收 TCP 输入缓冲区是 4 千字节。发送方发送前 4 千字节数据。接收方将其存储在缓冲区中。现在缓冲区满了。它通告一个大小为 0 的窗口，这意味着发送方应该停止发送数据。接收方应用从接收方 TCP 输入缓冲区中读取第一个字节的数据。现在接收缓冲区有 1 字节空间。接收 TCP 声明一个大小为 1 字节的窗口，这意味着迫不及待要发送数据的发送方 TCP 把这个通告看做是一个好消息，并发送只携带 1 字节数据的段。这个流程将继续。1 字节数据被消耗并且携带 1 字节数据的段被发送。我们再一次遇到了效率问题以及糊涂窗口综合征。

此处提出两种解决方法来防止由消耗数据速率低于数据到达速率而产生的糊涂窗口综合征。第

一种方法是 Clark 解决方法（Clark's solution），即数据一到达就发送确认，但是声明一个大小为 0 的窗口，直到有足够的空间容纳最大段，或者至少半个接收缓冲区是空的。第二种方法是延迟发送确认。这意味着当段到达，它不立即确认。接收方延迟确认，直到在确认那些到达段之前接收缓冲区中有适当的空间。延迟确认防止发送 TCP 滑动窗口。在发送 TCP 发送窗口里的数据之后，它停止。这样消除了症状。

延迟确认有另外一个优势：它减少通信量。接收方不必确认每个段。然而，它也有缺点，那就是延迟确认可能导致发送方不必要地重传那些未确认段。

协议平衡了优势和劣势。现在它定义了确认不能延迟超过 500ms。

3.4.8　差错控制

TCP 是一个可靠的传输层协议。这意味着将数据流传递给 TCP 的应用程序依靠 TCP 将整个数据流传递给另一端的应用程序，并且是按序的、无差错的、没有任何一部分丢失或重复的。

TCP 使用差错控制提供可靠性。差错控制包括用于检测并重发损坏段的机制、用于重发丢失的段的机制、用于存储失序的段直到丢失段到达的机制，以及检测并丢弃重复段的机制。TCP 中的差错检测和纠正通过三种简单工具来完成：校验和、确认和超时。

校验和

每个段都包括校验和字段，用来检查损坏的段。如果段被损坏，它将被目的端 TCP 丢弃，并被认为是丢失了。TCP 在每段中强制使用一个 16 位的校验和。在第 5 章我们将会讨论校验和的计算。

确认

TCP 使用确认方法来证实收到了数据段。不携带数据但占用序号的一些控制段也要确认，但 ACK 段是不确认的。

> ACK 段不占用序号，它不需要确认。

确认类型

在过去，TCP 只使用一种类型确认：累积确认。现在，一些 TCP 实现也使用选择性确认。

累积确认（ACK）　最初的 TCP 被设计成累积确认接收段。接收方通告下一个预期接收的字节，忽略所有失序段。这有时称为*积极累积确认*（positive cumulative acknowledgment）或 ACK。*积极*这个词表示不为丢弃、丢失或被破坏的段提供反馈。TCP 头部的 32 位 ACK 字段用来累积确认，且只有当 ACK 标志位置为 1 时才有效。

选择性确认（SACK）　越来越多的实现加入了另外一种称为*选择性确认*（selective acknowledgment）或 SACK 的确认类型。SACK 并不替代 ACK，但它向发送方报告额外的信息。SACK 报告失序字节块，也报告重复字节块即接收了一次以上的字节块。然而，因为 TCP 头部没有为加入这种类型的信息做准备，SACK 以一种 TCP 头部末端选项的形式实现。当我们在本书的网站讨论 TCP 选项的时候，我们会讨论这个新特性。

产生确认

接收方什么时候产生确认？在 TCP 的发展历程中，定义了很多规则也使用了很多种实现。我们给出最常见的规则。规则的顺序不代表其重要性。

1．当终端 A 向终端 B 发送一个数据段时，它必须包含（捎带）一个确认，这个确认给出下一个期待接收的序号。这个规则降低了所需段的数量，因此减少了通信量。

2．当接收方没有数据要发送并接收到一个有序段（带有预期序号），并且之前的段已经被确认，那么接收方延迟发送 ACK 段直到另一个段到达，或者过一段时间之后（通常 500ms）。换言之，如果只有一个未完成的有序段，那么接收方需要延迟发送 ACK 段。这个规则减少了 ACK 段。

3．当一个带有接收方预期序号的段到达，且之前一个有序段未被确认，接收方立即发送一个

ACK 段。换言之，任何时候不能多于两个有序的未确认段存在。这防止了不必要的重传，它可能引起网络拥塞。

4．当一个失序段到达，且它的序号大于预期，那么接收方立即发送一个 ACK 段，声明下一个预期段的序号。这导致了丢失段的快速重传（fast retransmission）（后面会讨论）。

5．当一个丢失段到达，接收方发送一个 ACK 段，声明下一个预期序号。这通知接收方被报告丢失的段已经到达。

6．如果重复段到达，接收方丢弃段，但是立即发送一个确认指出下一个预期的有序段。这个方法解决了当 ACK 段丢失时的一些问题。

重传

差错控制机制的核心是段的重传。当一个段被发送，它就被储存在一个队列中直到被确认。当重传计时器超时或当发送方接收到对队列中的第一个段的三次重复 ACK 时，就重传这个段。

RTO 之后重传

发送方 TCP 为每个连接维护一个**重传超时**（retransmission time-out，RTO）。当计时器到时，即超时，TCP 重发队列前面的段（具有最小序号的段）并重启计时器。注意，我们再次假设 $S_f < S_n$。我们将在后面看到 RTO 的数值在 TCP 中是动态的，并根据段的往返时间（round-trip time，RTT）进行更新。RTT 是一个段到达目的端并接收到一个确认所需要的时间。

三次重复 ACK 段之后重传

如果 RTO 的值不是很大，那么之前关于段重传的规则就足够了。当今，大多数实现遵循三次重复 ACK 规则，立即重发缺少的段，这种方法是通过允许发送方不等待超时而重传加速了因特网中的服务。这个特性称为**快速重传**（fast retransmission）。在这个版本中，如果三个对同一个段的重复确认（即一个原始 ACK 加上三个完全相同的副本）到达，那么下一个段就被重传而不必等待超时。我们在本章后面回到这个特性。

失序段

当今的 TCP 实现不丢弃失序段。它们暂时存储这些失序段，并将其标记成失序段直到缺失的段到达。然而，注意，失序段不传递给进程。TCP 确保数据按序传递给进程。

> 数据可以失序到达，并被接收的 TCP 暂时存储。但是 TCP 确保传递给进程的段是非失序的。

TCP 中数据传输的有限状态机

TCP 中的数据传输与选择性重复协议中的数据传输接近，与 GBN 有些许相似。因为 TCP 接收失序段，TCP 可以被认为行为上更像 SR 协议，但是因为原始确认是累积的，它看起来像 GBN。然而，如果 TCP 实现使用 SACK，TCP 最接近 SR。

> 以选择性重复协议为模型对 TCP 进行描述效果最好。

发送方有限状态机

让我们给出一个 TCP 协议发送方的简化有限状态机，它与我们在 SR 协议中讨论的相似，但是对于 TCP 有一些改变。我们假设通信是单向的，且发送的段被 ACK 段确认。我们也暂时忽略选择性确认和拥塞控制。图 3-59 给出了发送方简化的有限状态机。注意有限状态机是初步的；它不包含糊涂窗口综合征（Nagle 算法）或窗口关闭。它定义了单向通信，忽略所有影响双向通信的问题。

图 3-59 中的有限状态机与我们讨论的 SR 协议有一些不同。一个区别是快速重传（三次重复 ACK）。另一个区别是基于 rwnd 数值进行窗口大小调整（暂时忽略拥塞控制）。

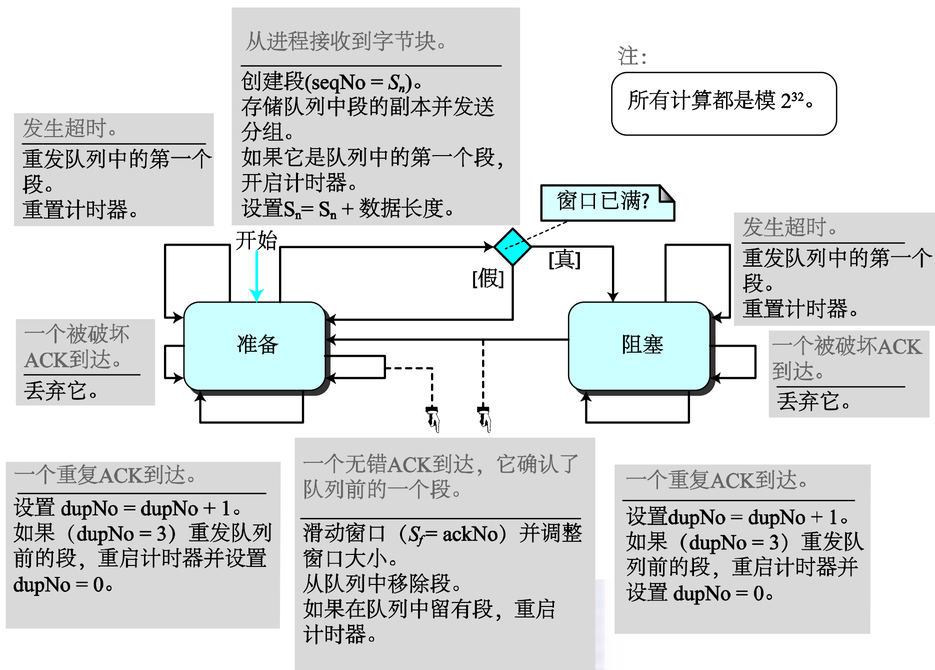


图 3-59 TCP 发送方的简化有限状态机

接收方有限状态机

让我们给出一个 TCP 协议接收方的简化有限状态机，它与我们在 SR 协议中讨论的相似，但是对于 TCP 有一些改变。我们假设通信是单向的，且使用 ACK 段确认段。我们也暂时忽略选择性确认和拥塞控制。图 3-60 给出了发送方简化的有限状态机。注意，我们忽略了一些问题，如糊涂窗口综合征（Clark 解决方法）与窗口关闭。

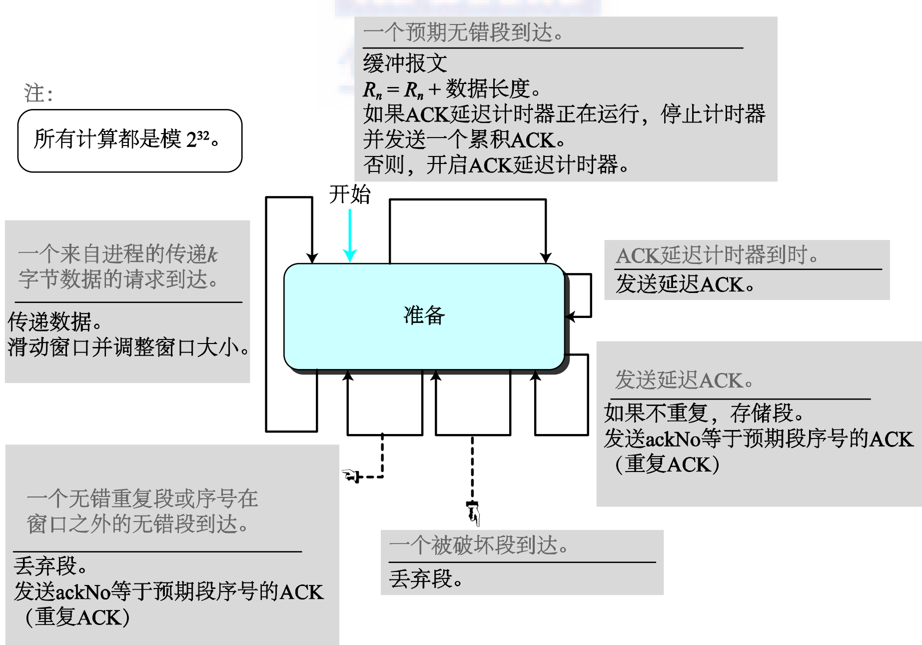


图 3-60 TCP 接收方简化有限状态机

这个有限状态机和我们在 SR 协议中讨论的有一些不同。一个区别是 ACK 在单向通信中延迟。另一个区别是发送重复 ACK 允许发送方实现快速重传策略。

我们也需要强调接收方的双向有限状态机不像 SR 中那么简单；我们需要考虑一些策略，比如，如果接收方有数据返回，那么立即发送一个 ACK。

某些场景

在本节，我们给出 TCP 操作时某些场景的例子。在这些场景中，用长方形表示段。如果段携带数据，我们显示字节序号的范围和确认字段的值。如果段仅是一个确认，我们仅用小方框中的确认号表示。

正常操作

第一种场景表示了两个系统之间的双向数据传输，如图 3-61 所示。客户 TCP 发送一个段，而服务器 TCP 发送三个段。图 3-61 表示了应用于每个确认的规则。在服务器端，只应用规则 1。如果有数据要发送，那么该段显示预期接收的下一个字节序号。当客户端接收到来自服务器的第一个段，而且它没有更多的数据要发送，那么它仅仅需要发送一个 ACK 段。但是，根据规则 2，该确认段需要延迟 500ms 以观察是否还有更多的段到达。当 ACK 延迟计时器到时，它出发一个确认。这样做是因为客户端不知道是否还有其他的段到来；它不能永远延迟确认。当下一个段到达时，启动另一个确认计时器。但是在它超时之前，第三个段到达，第三个段的到达触发另一个基于规则 3 的确认。我们没有给出 RTO 计时器，因为没有段丢失或延迟。我们仅仅假设 RTO 计时器起了作用。

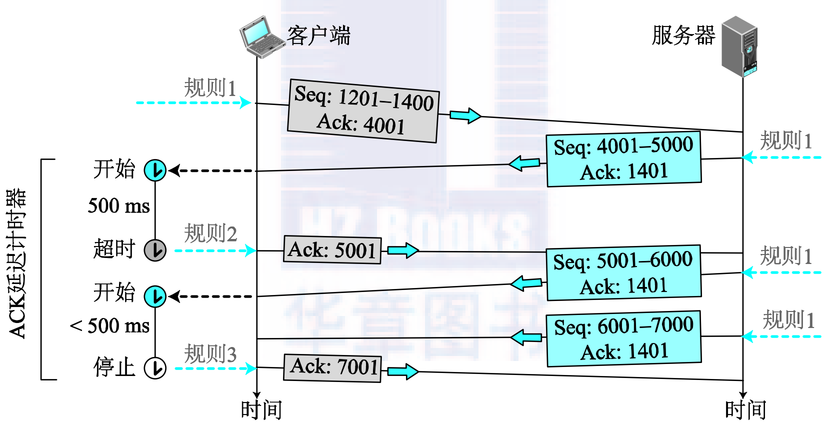


图 3-61　正常操作

丢失的段

在这种场景下，我们说明了当一个段丢失或损坏时将发生什么。接收方对丢失的段和损坏的段用相同的方法处理。丢失的段是在网络中的某处丢失的，损坏的段是被接收方本身丢弃的。图 3-62 给出了一种情况，一个段丢失（或许由于拥塞被在网络中的某个路由器丢弃掉）。

我们假定数据传输是单向的：一方发送，另一方接收。在我们的场景下，发送方发送段 1 和 2，这两个段立即被一个 ACK 确认（规则 3）。但是，段 3 被丢失了，接收方接收到段 4，发生了失序。接收方将数据存储在它的缓冲区的段中，但留出一个间隙指明数据中存在不连续性。接收方立即给发送方发送一个确认，表示了它预期的下一个字节。注意，接收方存储从 801 到 900 的字节，但是在这个间隙被填充之前不将这些字节传递给应用程序。

接收方 TCP 仅将有序的数据传递给进程。

发送方 TCP 在整个连接周期保持一个 RTO 计时器。当第三个段超时，发送方 TCP 重发段 3，

这次段 3 到达且被确认（规则 5）。

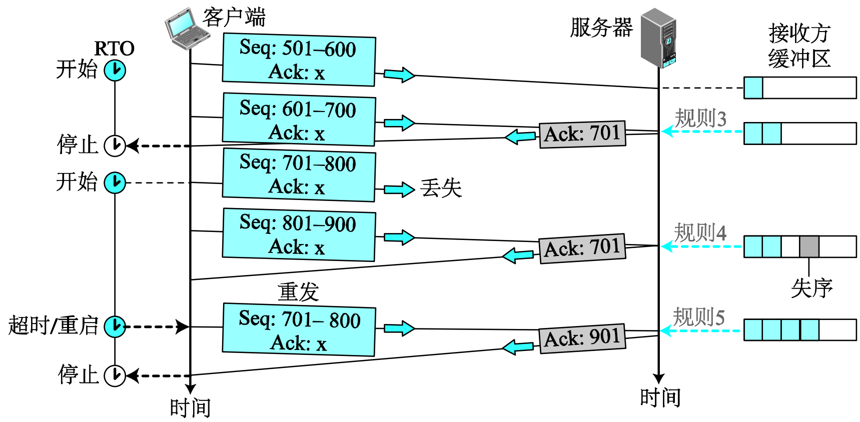


图 3-62　丢失的段

快速重传

在这个场景下，我们要给出快速重传。除了 RTO 具有较大的值外，该情况与第二种情况相同（见图 3-63）。

接收方每次接收到一个后续的段，它就触发一个确认（规则 4）。发送方接收四个具有相同值的确认（三个是重复的）。尽管计时器没有到时，但是快速重传要求立即重发段 3，该段是所有这些确认所预期的。在重发这个段之后，计时器被重启。

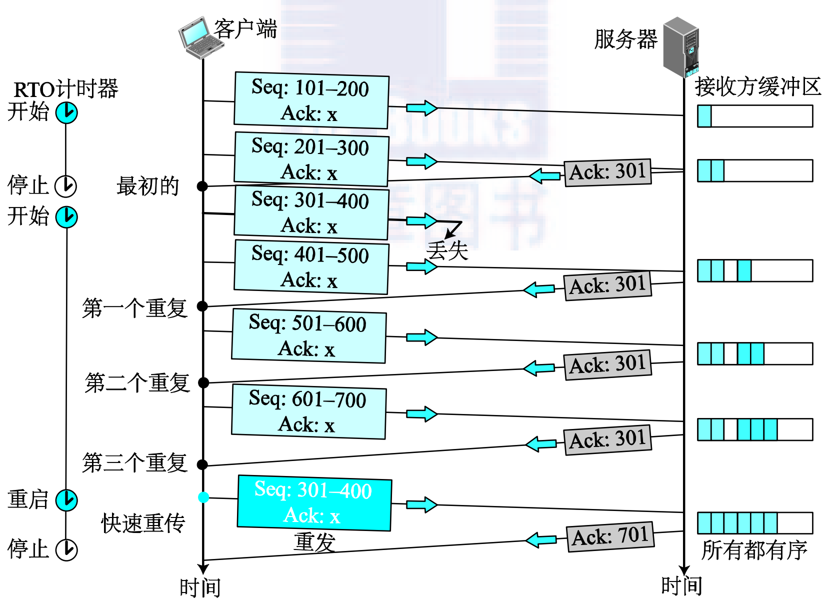


图 3-63　快速重传

延迟段

第四个场景描述了一个延迟段。TCP 使用 IP 服务，是一个无连接协议。每个 IP 数据报封装一个 TCP 段，它可能通过一条不同的路径以不同的延迟到达最终目的地。因此 TCP 段可能延迟。延迟段有时可能超时并被重传。如果延迟段在它被重传后到达，那么它就被认为是重复段并被丢弃。

重复段

例如，当一个段延迟并被接收方作为丢失报文时，一个重复段可以被创建。处理重复段对目的 TCP 是一个简单的过程。目的 TCP 预期连续字节流。当段到达，且其序号等于已经被接收且存储的段序号，那么它就被丢弃。ACK 被发送，其中 ackNo 定义了预期段。

自动更正丢失 ACK

这个场景给出一种情况，在丢失确认中的信息被包含在下一个确认中，这是使用累积确认的一个关键优势。图 3-64 给出了数据接收方发送的丢失确认。在 TCP 确认机制中，丢失确认甚至不会被源 TCP 通知。TCP 使用累积确认。我们可以说下一个确认自动更正之前的确认丢失。

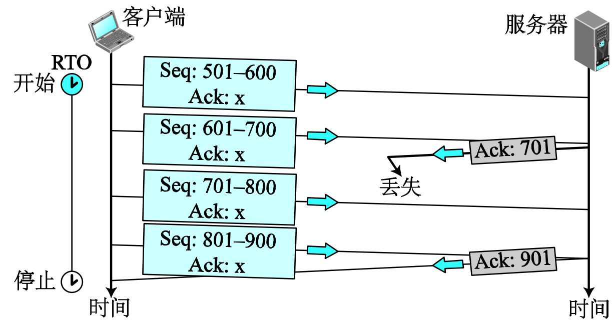


图 3-64　丢失确认

被重发段更正的丢失确认

图 3-65 给出了确认丢失的场景。

如果下一个确认延迟了很长时间或不存在确认（丢失的确认是上一次被发送的），RTO 计时器触发更正。这会造成重复段。当接收方接收到一个重复段，它就立即丢弃并重发上一个 ACK 通知发送方段已经被接收。

注意，尽管两个段没有被确认，但是只有一个段被重传。当发送方接收到重传 ACK，它知道双方是安全和完整的，因为确认是累积的。

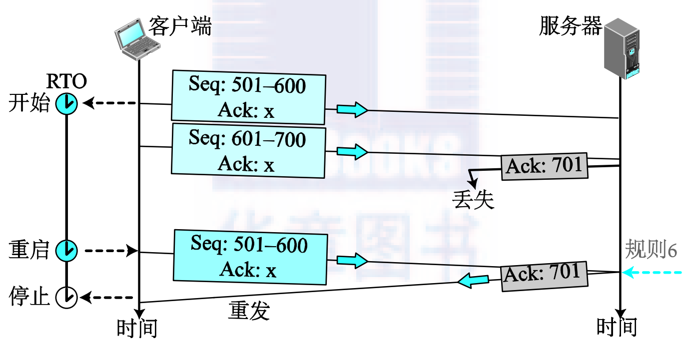


图 3-65　被重发段更正的丢失确认

丢失确认造成的死锁

存在一种情况，其中丢失确认可能导致系统死锁。这种情况中接收方发送 rwnd 设置为 0 的确认，并要求发送方临时关闭它的窗口。在一段时间后，接收方想要去掉限制；然而，如果它没有数据要发送，它就发送一个 ACK 段并以 rwnd 非零值去除限制。如果这个确认丢失了，可能产生一个问题。发送方正在等待声明非零 rwnd 的确认。接收方认为发送方已经接收到了这个确认，而且正在等待数据。这种情况称为**死锁**（deadlock）；每个终端等待来自另一端的响应，且任何事情都没有发生。重传计时器没有被设置。为了防止死锁，坚持计时器被设计出来，我们在本章稍后会研究到。

如果处理不当，丢失确认可能造成死锁，

3.4.9　TCP 拥塞控制

TCP 使用两种不同的策略来处理网络中的拥塞。我们在本节描述这些策略。

拥塞窗口

当我们讨论 TCP 中的流量控制，我们曾提到过接收方使用 rwnd 的数值来控制发送窗口，它在每个沿相反方向传递的段中被通告。使用这个策略保证了接收窗口不会被接收字节溢出（没有终端拥塞）。然而，这不意味着中间缓冲区、路由器中的缓冲区不会变得拥塞。路由器可能从不止一个发送端接收数据。无论路由器的缓冲多大，它都可能被数据淹没，这导致特定 TCP 发送方丢弃某些段。换言之，在另一端不存在拥塞，但是可能在中间存在拥塞。TCP 需要担心中间的拥塞，因为很多丢失段可能导致差错控制。更多的段丢失意味着再次重发相同的段，导致拥塞更严重，并且最终导致通信崩溃。

TCP 是使用 IP 服务的端到端协议。路由器中的拥塞是在 IP 域内，并且应该由 IP 解决。然而，正如我们在第 4 章讨论的，IP 是一个没有拥塞控制的简单协议。TCP 自身需要为这个问题负责。

TCP 不能忽略网络中的拥塞问题；它不能过分激进地向网络中发送段。正如之前提到的，这样激进的结果只能伤害 TCP 自身。TCP 也不能过于保守，每个时间间隔只发送少量的段，因为这意味着没有利用好网络可用带宽。TCP 需要定义当没有拥塞时的加速数据传输策略以及当检测到拥塞时的减速策略。

TCP 使用称为*拥塞窗口*（congestion window，cwnd）的变量来控制段的发送数量，这个变量的值由网络中的拥塞情况所控制（我们马上就会解释）。cwnd 变量和 rwnd 变量一起定义了 TCP 中的发送窗口大小。第一个变量与中间的拥塞相关（网络）；第二个变量与终端的拥塞相关。实际窗口的大小是这两者中的最小值。

$$实际窗口大小 = \text{minimum (rwnd, cwnd)}$$

拥塞检测

在讨论 cwnd 的值如何被设置和改变之前，我们需要描述 TCP 发送方如何检测到网络中可能存在的拥塞。TCP 发送方使用两个事件作为网络中拥塞的标志：超时和接收到三次重复 ACK。

第一个是*超时*（time-out）。如果一个 TCP 发送方在超时之前没有接收到对于某个段或某些段的 ACK，那么它就假设相应段或相应那些段丢失了，并且丢失是拥塞引起的。

另一个事件是接收到三次重复 ACK（四个带有相同确认号的 ACK）。回忆当 TCP 接收方发送一个重复 ACK，这是段已经被延迟的信号，但是发送三次重复 ACK 是丢失段的标志，这可能是由于网络拥塞造成的。然而，在三次重复 ACK 的情况下拥塞的严重程度低于超时情况。当接收方发送三次重复 ACK 时，这意味着一个段丢失，但是三个段已经被接收到。网络或者轻微拥塞或者从已经从拥塞中恢复。

我们将在稍后给出一个较早版本的 TCP，称为 Taho TCP，它处理两种事件（超时和三次重复 ACK）是相似的，但是之后的 TCP 版本，称为 Reno TCP，处理这两种事件就不同了。

TCP 拥塞中非常有趣的一点是，TCP 发送方只使用一种反馈从另一端来检测拥塞：即 ACK。没有周期性地、及时地接收到 ACK，这导致超时，是严重拥塞的标志；接到三次重复 ACK 是网络中轻微拥塞的标志。

拥塞策略

TCP 处理拥塞的一般策略基于三个算法：慢启动、拥塞避免和快速恢复。在给出在连接中 TCP 如何从一种算法转到另一种算法之前，我们先讨论每一种算法。

慢启动：指数增加

慢启动（slow-start）算法是基于拥塞窗口大小（cwnd）的思想，它以最大段长度（MSS）开始，但是每当一个确认到达时它只增加一个 MSS。如我们之前讨论的，MSS 是连接建立期间由同名的最大段长度选项协商产生的值。

这个算法的名字有些误导：算法启动慢，但它是以指数增长的。为了表示这个思想，让我们看

图 3-66。我们假设 rwnd 比 cwnd 大得多，因此发送窗口大小永远等于 cwnd。我们也假设每个段是同长度的，并携带 MSS 字节。为了简单起见，我们也忽略延迟 ACK 策略并假设每个段单独被确认。

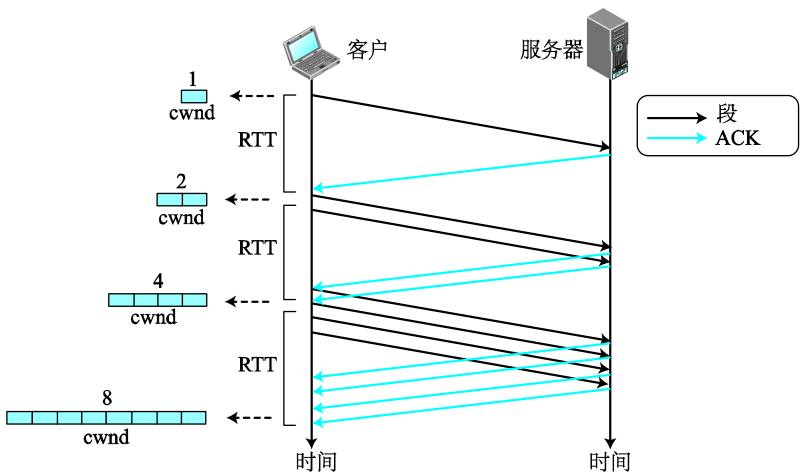


图 3-66　慢启动，指数增加

发送方以 cwnd = 1 开始。这意味着发送方仅能发送一个段。当第一个 ACK 到达后，被确认的段被从窗口中清除，这意味着现在窗口中有一个空段槽。拥塞窗口的大小也增加 1，因为收到确认标志着网络中没有拥塞。窗口的大小现在是 2。在发送两个段并接收到两个独立的确认之后，现在拥塞窗口的大小是 4，依此类推。换言之，在这个算法中拥塞窗口的大小是到达 ACK 数量的函数，可由下式决定。

如果一个 ACK 到达，cwnd = cwnd + 1。

如果我们按照往返时间（RTT）观察 cwnd 的大小，那么我们发现其增长速率是指数的，这是一个非常激进的方法：

开始 → cwnd = 1 → 2^0

第一个 RTT 之后 → cwnd = cwnd + **1** = **1** + **1** = **2** → $\mathbf{2^1}$

第二个 RTT 之后 → cwnd = cwnd + **2** = **2** + **2** = **4** → $\mathbf{2^2}$

第三个 RTT 之后 → cwnd = cwnd + **4** = **4** + **4** = **8** → $\mathbf{2^3}$

慢启动不能一直继续下去。肯定存在一个停止该阶段的阈值。发送方保存一个称为 ssthresh（slow-start threshold，慢启动阈值）的变量。当窗口中的字节达到这个阈值时，慢启动停止且下一个阶段开始。

在慢启动算法中，拥塞窗口大小按指数规律增长直到到达阈值。

然而，我们已经提到慢启动策略在延迟确认情况下更慢。请记住，对于每个 ACK，cwnd 值增加 1。因此，如果两个段被累积确认，cwnd 大小只增加 1 不是 2。增长仍是指数的，但是它不是 2 的幂。对于确认了两个段的 ACK，它是 1.5 的幂。

拥塞避免：加性增加

如果我们继续慢启动算法，那么拥塞窗口大小按指数规律增大。为了在拥塞发生之前避免拥塞，必须降低指数增长的速度。TCP 定义了另一个算法，称为**拥塞避免**（congestion avoidance），这个算法是加性增加 cwnd 而不是指数增加。当拥塞窗口的大小到达慢启动的阈值时，这种情况下 cwnd = *i*，慢启动阶段停止且加性增加阶段开始。在这个算法中，每次整个“窗口”的所有段都被确认（一次传输），拥塞窗口才增加 1。窗口是 RTT 期间传输的段的数量。图 3-67 说明了这个概念。

发送方以 cwnd = 4 开始。这意味着发送方只能发送 4 个段。在 4 个 ACK 到达之后，被确认的段被从窗口中清除，这意味着现在窗口中有一个空闲段。拥塞窗口也增加 1。窗口大小现在为 5。在发送 5 个段并接收到 5 个确认之后，拥塞窗口大小变为 6。其余以此类推。换言之，这个算法中拥塞窗口的大小也是到达的 ACK 数量的方程，它由下式决定：

如果一个 ACK 到达，cwnd = cwnd + (1/cwnd)。

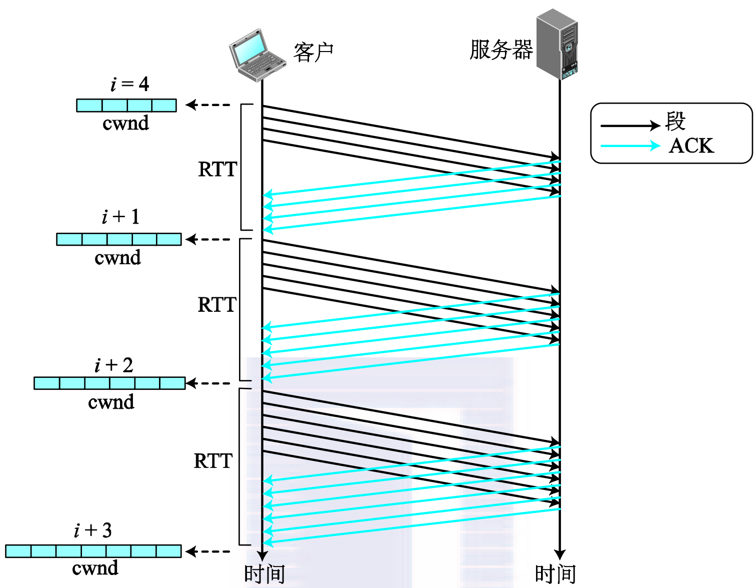


图　3-67

换言之，窗口大小每次只增加 MSS 的 1/cwnd（以字节为单位）。换言之，窗口中所有段都需要被确认，才能使窗口增加 1MSS 字节。

如果我们按照往返时间（RTT）观察 cwnd 的大小，那么我们会发现其增长速率以每次往返时间为单位是线性的，这比慢启动方法保守多了。

开始	→	cwnd = i
第一个 RTT 之后	→	cwnd = i + 1
第二个 RTT 之后	→	cwnd = i + 2
第三个 RTT 之后	→	cwnd = i + 3

在拥塞避免算法中，在检测到拥塞之前，拥塞窗口大小是加性增加的。

快速恢复　快速恢复（fast-recovery）算法在 TCP 中是可选的。旧版本的 TCP 不使用它，但是新版本使用。快速恢复开始于三次重复 ACK 到达，这被解释为网络的轻微阻塞。像拥塞避免一样，这个算法也是加性增加的，但是当一个重复 ACK 到达时（在三次重复 ACK 触发使用这个算法之后），它增加拥塞窗口的大小。我们可以说：

如果一个重复 ACK 到达，cwnd = cwnd + (1/cwnd)。

策略转换

我们讨论了 TCP 中的三种拥塞策略。现在的问题是何时使用这些策略，并且 TCP 何时从一种策略转换到另一种策略。为了回答这些问题，我们需要参照三种 TCP 版本：Taho TCP、Reno TCP 以及新 Reno TCP。

Taho TCP

早期的 TCP 称为 Taho TCP，它只使用两种拥塞策略算法：慢启动和拥塞避免。我们使用图 3-68 给出这个版本 TCP 的有限状态机。然而，需要提及的是，我们已经删除了一些琐碎行为，例如增加和重置重复 ACK 数量，这使得有限状态机不那么臃肿且更简单。

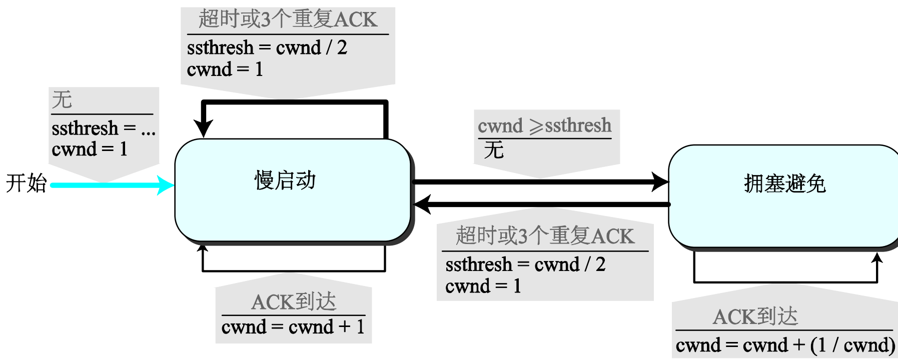


图 3-68　Taho TCP 有限状态机

Taho TCP 用相同的方式对待拥塞检测的两种情况，即超时和三次重复 ACK。在这个版本中，当连接建立，TCP 开始慢启动算法，并将变量 ssthresh 设置为之前协商好的数值（通常是 MSS 的倍数），将 cwnd 设置为 1MSS。在这种状况下，如前所述，每次 ACK 到达，拥塞窗口增加 1。我们知道这个策略很激进并且窗口是指数增加的，这可能导致拥塞。

如果检测到拥塞（发生超时或三次重复 ACK），TCP 立即中断这个激进的增长，并将阈值限定为当前 cwnd 的一半，重置拥塞窗口为 1，从而重启一个新的慢启动。换言之，不仅 TCP 从零开始，而且它还学到了如何调整阈值。如果当达到阈值时没有检测到拥塞，TCP 知道已经到达目标的顶点；它不应该继续以此速度增加。它进入拥塞避免状态并继续这种状态。

在拥塞避免状态，每当 ACK 数目等于当前已接收的窗口大小时，拥塞窗口大小增加 1。例如，如果现在窗口大小是 5MSS，在窗口大小变为 6MSS 之前，需要再接收 5 个 ACK。注意，在这种状态下，没有拥塞窗口大小的上限；除非检测到拥塞，拥塞窗口的保守加性增加将会继续，直到数据传输阶段结束。如果在这个状态下检测到拥塞，TCP 再次将 ssthresh 的值重置为 cwnd 的一半，并进入慢启动状态。

尽管这个版本的 TCP 中，ssthresh 的大小在每次拥塞检测中都在不断调整，但是这并不意味着它必然变得比之前的数值小。例如，当 TCP 处于拥塞避免阶段且 cwnd 是 20 时，如果初始 ssthresh 数值为 8MSS 且检测到拥塞，那么现在的新 ssthresh 的数值为 10，这意味着它增加了。

例 3.19　图 3-69 给出了 Taho TCP 中拥塞控制的例子。TCP 开始数据传输并将 ssthresh 设为雄心勃勃的 16MSS。TCP 开始慢启动（SS）阶段 cwnd = 1。拥塞窗口指数增长，但是在第三个 RTT 发生超时（在到达阈值之前）。TCP 假设网络中存在拥塞。它立即设置新的 ssthresh = 4MSS（当前 cwnd 的一半，cwnd 为 8）并开始一个新的慢启动（SA）状态，并令 cwnd = 1MSS。拥塞指数增长，直到它到达了新设置的阈值。现在 TCP 进入拥塞避免（CA）状态且拥塞窗口加性增加，直到它到达 cwnd = 12MSS。这时，三次重复 ACK 到达，这是网络拥塞的另一个象征。TCP 再次将 ssthresh 削减一半至 6MSS，并且开始一个新的慢启动（SS）状态。cwnd 的指数增长继续。在 RTT 15 之后，cwnd 的数值为 4MSS。在发送四个段并接收两个 ACK 后，窗口大小到达阈值（6）且 TCP 进入拥塞避免状态。现在数据传输继续处于拥塞避免（CA）状态直到 RTT 20 之后连接终止。

Reno TCP

一个新版 TCP，称为 Reno TCP，在拥塞控制有限状态机中加入了新的状态，即快速恢复状态。这个版本用不同的方法处理拥塞检测的两种情况，即超时和三次重复 ACK。在这个版本中，如果发生超时，TCP 进入慢启动状态（如果它已经处于该状态，则开始新的一轮）；另一方面，如果三

次重复 ACK 到达，则 TCP 进入快速恢复阶段，并且只要有更多的重复 ACK 到达，它就保持这种状态。快速恢复状态是一种介于慢启动和拥塞避免之间的状态。它像慢启动，其中 cwnd 以指数增长，但是 cwnd 以 ssthresh 加 3MSS（而不是 1）开始。当 TCP 进入快速恢复阶段，可能发生三种主要事件。如果重复 ACK 继续到达，那么 TCP 保持这种状态，但是 cwnd 呈指数增长。如果发生超时，TCP 假设网络中有真实的拥塞，并进入慢启动状态。如果一个新的（非重复）ACK 到达，TCP 进入拥塞避免阶段，但是将 cwnd 的大小减小到 ssthresh 值，好像三次重复 ACK 没有发生过一样，并且转换是从慢启动状态到拥塞避免状态。图 3-70 给出一个 Reno TCP 的简化有限状态机。我们再次去除了一些琐碎事件来简化图和讨论。

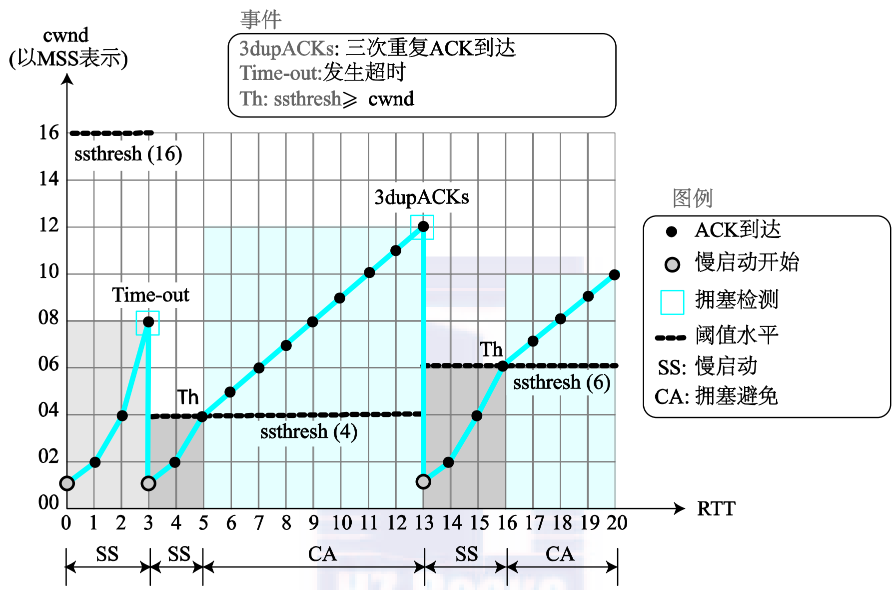


图 3-69　Taho TCP 例子

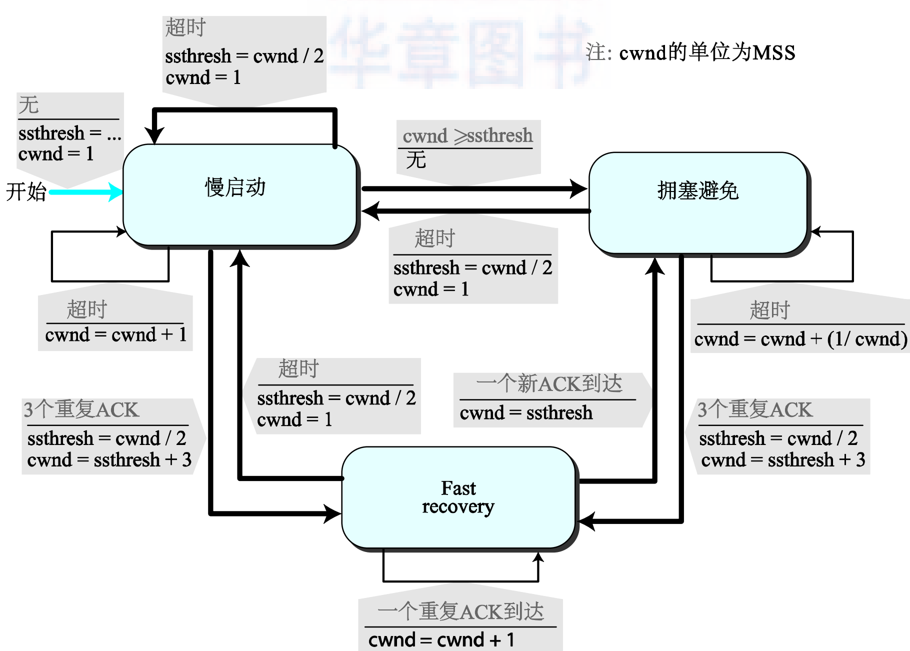


图 3-70　Reno TCP 有限状态机

例 3.20　图 3-71 给出了与图 3-69 相同的情况，但图 3-71 是在 Reno TCP 中。在三次重复 ACK 到达的 RTT 13 之前，拥塞窗口的变化都是相同的。此时，Reno TCP 将 ssthresh 降低到 6MSS（与 Taho TCP 相同），但是它将 cwnd 设置到一个更大的数值（ssthresh + 3 = 9MSS）而不是 1MSS。Reno TCP 现在进入快速恢复状态。我们假设有两个重复 ACK 在 RTT 15 到达，此处 cwnd 呈指数增长。此时，一个新的 ACK（不是重复的）到达，声明接收到丢失段。现在 Reno TCP 进入拥塞避免状态，但是首先将拥塞窗口减少到 6MSS（ssthresh 值），好像忽略整个快速重传状态并移动回之前的轨迹。

NewReno TCP

一个 TCP 的后期版本称为 NewReno TCP，它在 Reno TCP 基础上进行了额外优化。在这个版本中，当三次重复 ACK 到达时，TCP 检查在当前窗口中是否有一个以上的段丢失。当 TCP 接收到三次重复 ACK 时，它重传丢失段直到一个新的 ACK（非重复）到达。当检测到拥塞时，如果新的 ACK 定义了窗口的末端，TCP 可以确定只有一个段丢失。然而，如果 ACK 数定义了重传段和窗口末端之间的一个位置，那么被 ACK 定义的段也可能丢失了。NewReno TCP 重传这个段以避免接收到关于这个段的越来越多的重复 ACK。

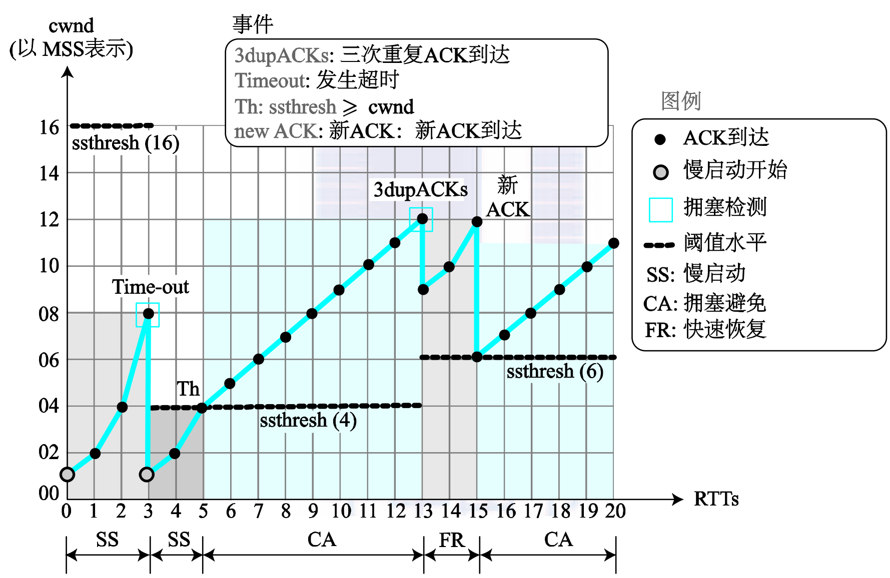


图 3-71　Reno TCP 例子

加性增加，乘性减少

在这三种 TCP 版本中，Reno 版本是如今最普遍的。已经观察到，在这个版本中，绝大多数情况下，通过观察三次重复 ACK，可以检测并处理拥塞。即使有一些超时事件，TCP 也会通过激进的指数增长从中恢复。换言之，在长时间的 TCP 连接中，如果我们忽略在快速恢复期间的慢启动状态和短暂指数增长，那么，当 ACK 到达（拥塞避免）时 TCP 拥塞窗口为 cwnd = cwnd + (1/cwnd)，并且当检测到拥塞时 cwnd = cwnd / 2，好像 SS 不存在且 FR 的长度减小为 0。第一个称为*加性增加*；第二个称为*乘性减少*。这意味着在经过初始慢启动状态后，拥塞窗口大小遵循锯齿样式，称为**加性增加，乘性减少**（additive increase，multiplicative decrease，AIMD），如图 3-72 所示。

TCP 吞吐量

TCP 的吞吐量是基于拥塞窗口的行为，如果 cwnd 是 RTT 的常数（平直直线）函数，那么吞吐量可以很容易计算出来。这个不实际的假设得出吞吐量 = cwnd / RTT。在这个假设中，在 RTT 时间内，TCP 发送一个 cwnd 字节的数据并接收到对它们的确认。TCP 的行为，如图 3-72 所示，

不是一条平直直线；它像是锯齿，有很多最大值和最小值。如果每个齿都完全相同，我们可以说吞吐量 = [(maximum + minimum) / 2] / RTT。然而，我们知道最大值是最小值的两倍，因为在每次拥塞检测中，cwnd 的数值被设为之前值的一半。因此吞吐量可以计算为：

$$吞吐量 = (0.75)\ W_{max} / RTT$$

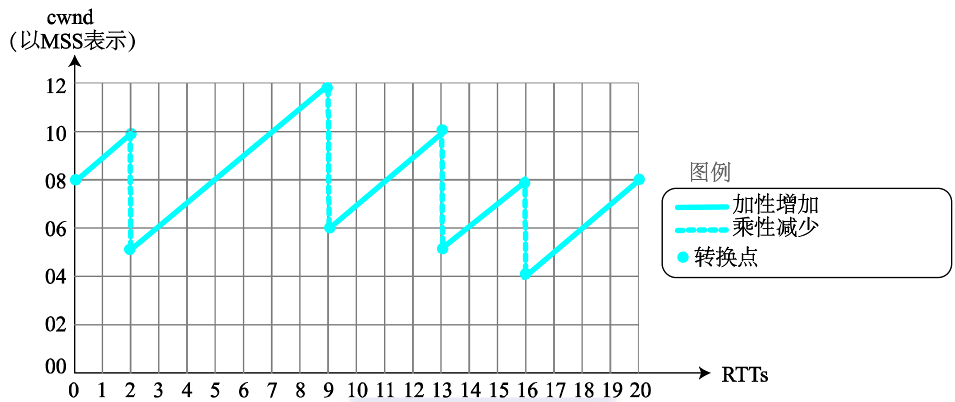


图 3-72 加性增加，乘性减少（AIMD）

其中 W_{max} 是当拥塞发生时窗口大小的平均值。

例 3.21 如果图 3-72 中 MSS = 10KB（千字节）且 RTT = 100ms，我们可以按如下计算吞吐量。

$W_{max} = (10 + 12 + 10 + 8 + 8) / 5 = 9.6$ MSS

吞吐量 $= (0.75\ W_{max} / RTT) = 0.75 \times 960$ kbps / 100 ms = 7.2 Mbps

3.4.10 TCP 计时器

为了更平稳地执行操作，绝大多数 TCP 实现使用至少四种计时器：重传、坚持、保活和时间等待计时器。

重传计时器

为了重传丢失的段，TCP 使用一种重传计时器（在整个连接期间）处理重传超时（RTO），即对一个段的确认等待时间。我们可以为重传计时器定义如下规则：

1. 当 TCP 发送位于发送队列前端中的段时，它开启计时器。
2. 当计时器到时，TCP 重发队列前端的第一个段并且重启计时器。
3. 当一个或多个段被累积确认，那么一个或多个段被从队列中清除。
4. 如果队列是空的，TCP 停止计时器。否则，TCP 重启计时器。

往返时间

为了计算重传超时（RTO），我们首先需要计算**往返时间**（Round-Trip Time，RTT）。然而，在 TCP 中计算 RTT 是一个复杂的过程，我们用一些例子一步一步进行解释。

- **测量 RTT**。我们需要得到发送一个段并接收它的确认所需的时间。这就是测量 RTT。我们需要记住，段以及它们的确认没有一一对应关系；几个段可能被一起确认。一个段的测量往返时间是段到达目的地并被确认所需的时间，尽管确认可能包含其他段。注意，在 TCP 中，任何时候只有一个 RTT 测量可以进行。这就意味着，如果 RTT 测量开始，直到 RTT 数值完成，不能开始其他测量。我们使用符号 RTT_M 表示测量 RTT。

  在 TCP 中，任何时候只能进行一个 RTT 测量。

- **平滑 RTT**。测量 RTT，RTT_M，可能在每个往返中改变。如今的因特网中波动很大，以至于

一次测量不能用于重传超时。绝大多数实现使用平滑 RTT，称为 RTT_S，这是 RTT_M 和前一个 RTT_S 的加权平均数，如下所示：

初始 → 无数值

在第一次测量后 → $\mathbf{RTT_S = RTT_M}$

在每次测量后 → $\mathbf{RTT_S = (1-\alpha)\ RTT_S + \alpha \times RTT_M}$

α的值依赖于实现，但是它通常被设置为 1/8。换言之，新 RTT_S 被计算成 7/8 的旧 RTT_S 加 1/8 的当前 RTT_M。

- **RTT 偏差**。绝大多数实现不仅仅使用 RTT_S；它们也计算称为 RTT_D 的 RTT 偏差，计算方法基于 RTT_S 和 RTT_M，使用如下方程（β值也是依赖实现的，但是通常设置为 1/4）：

初始 → 无数值

在第一次测量后 → $\mathbf{RTT_D = RTT_M /2}$

在每次测量后 → $\mathbf{RTT_D = (1-\beta)\ RTT_D + \beta \times | RTT_S - RTT_M |}$

重传超时（RTO） RTO 的值基于平滑往返时间以及它的偏差。绝大多数实现使用如下公式来计算 RTO：

最初 → 初始数值

在每次测量后 → $\mathbf{RTO = RTT_S + 4 \times RTT_D}$

换言之，将运行中 RTT_S 的平滑平均数加上 4 倍运行中 RTT_D 的平滑平均数（通常是一个很小的值）。

例 3.22 让我们来给出一个假想例子。图 3-73 给出连接的部分。图中给出连接建立和部分数据传输阶段。

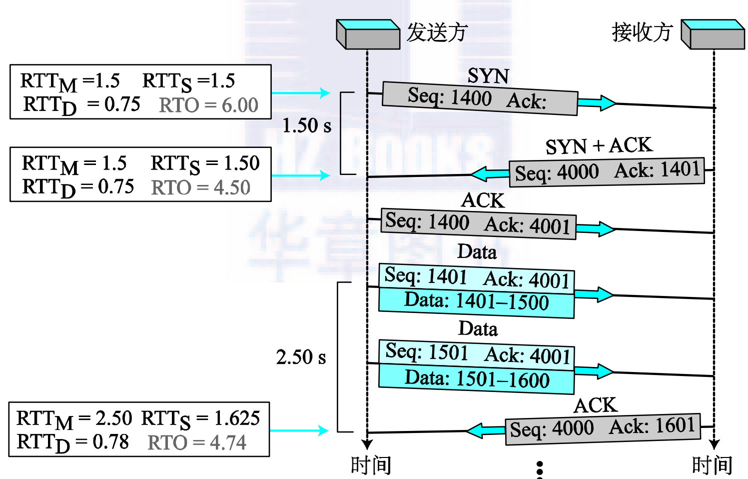


图 3-73 例 3.22

1. 当 SYN 段被发送，RTT_M、RTT_S 和 RTT_D 没有数值。RTO 的数值被设置为 6.00 秒。此时，以下给出了这些变量的数值：

$\mathbf{RTO = 6}$

2. 当 SYN + ACK 段到达，RTT_M 被测量且等于 1.5 秒。以下给出这些变量的数值：

$\mathbf{RTT_M = 1.5}$

$\mathbf{RTT_S = 1.5}$

$\mathbf{RTT_D = (1.5)/2 = 0.75}$

$\mathbf{RTO = 1.5 + 4 \times 0.75 = 4.5}$

3. 当第一个数据段被发送，一个新 RTT 测量开始。注意，当发送方发送一个 ACK 段，发送方不开始 RTT 测量，因为它并不消耗序号且没有超时。不对第二个数据段进行 RTT 测量，因为测量已经在进行。上一个 ACK 段的到达被用来计算下一个 RTT_M 数值。尽管上一个 ACK 段确认数据段（累积），它的到来完成了第一个段的 RTT_M 数值。这些变量的数值如下所示。

$\mathbf{RTT_M = 2.5}$

$\mathbf{RTT_S = (7/8) \times (1.5) + (1/8) \times (2.5) = 1.625}$

$\mathbf{RTT_D = (3/4) \times (0.75) + (1/4) \times |1.625 - 2.5| = 0.78}$

$\mathbf{RTO = 1.625 + 4 \times (0.78) = 4.74}$

Karn 算法

假设在重传超时期间段没有被确认，那么它应该被重传。当发送方 TCP 接收到这个段的确认，它不知道这个确认针对的是原始段的还是重传段。新 RTT 值基于段的离开。然而，如果原始段丢失且确认是针对重传段的，当前 RTT 值必须在段被重传的时候计算。这个歧义被 Karn 解决。**Karn 算法**很简单。不要考虑 RTT 计算中的重传段的往返时间。直到你发送一个段并接收一个段而不必重传时，才更新 RTT 数值。

> TCP 并不考虑新 RTO 计算中重传段的 RTT。

指数后退

如果发生重传，RTO 的值是多少？绝大多数 TCP 实现使用指数退后策略。RTO 的数值再每次重传中翻倍。因此如果段被重传一次，数值是两倍 RTO。如果被重传两次，数值是四倍 RTO，等等。

例 3.23　图 3-74 是之前例子的继续。这里应用了重传和 Karn 算法。

图中第一个段被发送，但是丢失了，RTO 计时器在 4.74 秒后到时。段被重传且计时器被设置为 9.48，是 RTO 数值的两倍。这次 ACK 在超时前被接收。我们等待，直到我们发送一个新的段并在重新计算 RTO 之前接收到针对它的 ACK（Karn 算法）。

坚持计数器

为了处理 0 窗口大小通告，TCP 需要另一个计时器。如果接收 TCP 声明了 0 窗口大小，那么发送 TCP 停止传输段，直到接收 TCP 发送一个 ACK 段声明非零的窗口大小。这个 ACK 段可能丢失。如果这个确认丢失了，接收 TCP 认为它已经完成了工作并等待发送方 TCP 发送更多的段。对于一个只包含确认的段不存在重传计时器。发送方 TCP 没有接收到确认，并且等待另一个 TCP 发送确认通告窗口的大小。两端 TCP 可能继续相互等待直到永远（死锁）。

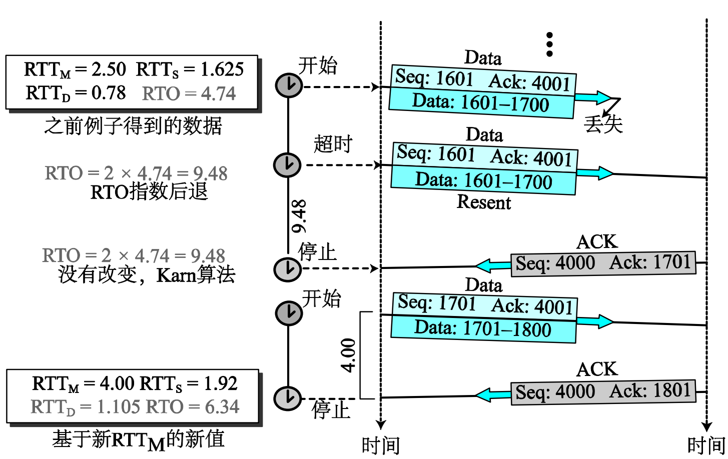


图 3-74　例 3.23

为了更正死锁。TCP 为每个连接使用**坚持计时器**（persistence timer）。当发送方 TCP 接收一个窗口大小为 0 的确认时，开启坚持计时器。当坚持计时器到时，发送方 TCP 发送一个特殊的段，称为探测（probe）。这个段只包含一字节的新数据。它有一个序号，但是序号从不被确认；在为剩余的数据计算序号时它甚至被忽略。探测报文引发接收方 TCP 重发确认。

坚持计时器的值被设置为重传时间。然而，如果一个响应没有被从接收方接收到，另一个探测段就会被发送且坚持计时器的数值会被加倍重置。发送方继续发送探测段并加倍重置坚持计时器数值，直到这个数值到达阈值（通常为 60s）。这之后，发送方每 60 秒发送一个探测段，直到窗口重新打开。

保活计时器

保活计时器（keepalive timer）通常在某些实现中使用，来防止两个 TCP 之间的长期空闲连接。假设有一个客户与一个服务器建立了连接，传输了一些数据并进入沉默状态。或许客户已经瘫痪。在这种情况下，连接会永远保持打开。

为了消除这种情况，绝大多数实现给服务器配备保活计时器。每当服务器从客户收到一次数据，就重置计时器。超时时间通常是 2 小时。如果服务器在 2 小时内没有收到客户数据，那么它发送一个探测段。如果每 75 秒钟发送一个探测段，一共发送 10 个探测段之后仍无客户响应，那么服务器就认为客户端出现了故障，并终止这个连接。

时间等待计时器

时间等待（TIME-WAIT）（2MSL）计时器在连接终止期间使用。最大报文段寿命（maximum segment life time，MSL）是任意报文在被丢弃前在网络中的存在时间。具体实现需要为 MSL 选择数值。通常为 30 秒、1 分钟甚至 2 分钟。当 TCP 执行主动关闭并发送最后一个 ACK 时，使用 2MSL 计时器。连接必须保持 2MSL 的时间，从而允许 TCP 重发最后一个 ACK，以防 ACK 丢失。这要求另一端的 RTO 计时器超时且新 FIN 和 ACK 段被重发。

3.4.11　选项

TCP 头部有多达 40 字节选项信息。选项向目的端传递额外信息或调整其他选项。这些选项包含在本书网站以供未来参考。

TCP 选项在本书网站上进行讨论。

3.5　章末资料

推荐读物

想要得到本章讨论主题的更多细节，我们推荐如下书籍和 RFC。

书籍

一些书籍论述了传输层协议相关内容。在本书末列出了方括号中的参考资料：我们尤其推荐[Com 06]、[Pet & Dav 03]、[Gar & Wid 04]、[Far 04]、[Tan 03]以及[Sta 04]。

RFC

与 UDP 相关的主要 RFC 是 RFC 768。一些 RFC 讨论 TCP 协议，包括 RFC 793、RFC 813、RFC 879、RFC 889、RFC 896、RFC 1122、RFC 1975、RFC 1987、RFC 1988、RFC 1993、RFC 2018、RFC 2581、RFC 3168 以及 RFC 3782。

小结

传输层协议的主要职责是提供进程到进程通信。为了定义进程，我们需要端口号。客户程序使用临时端口号定义自身。服务器使用熟知端口号定义自身。为了从一个进程向另一个进程发送报文，

传输层协议对报文进行封装和解封装。源端的传输层执行多路复用；目的端的传输层执行多路分解。流量控制平衡生产者和消费者之间的数据项交换。传输层协议可以提供两种类型的服务：无连接和面向连接。在无连接服务中，发送方向接收方发送分组，而没有连接建立。在面向连接服务中，客户和服务器首先需要在它们之间建立连接。

在本章中，我们已经讨论过很多常见的传输层协议。停止-等待协议提供了流量和差错控制，但是这是不够的。回退 *N* 帧协议是停止-等待协议的更高效版本并且使用流水线。选择性重复协议，一个回退 *N* 帧协议的修改版，更适合于处理分组丢失。所有这些协议都可以使用捎带实现双向通信。

UDP 是一个传输协议，它创建了进程到进程的通信。UDP（大多）是不可靠的无连接协议，它需要很小的开销并提供快速传递。UDP 分组称为用户数据报。

传输控制协议（TCP）是 TCP/IP 协议簇中的另一个传输层协议。TCP 提供进程到进程、全双工的面向连接服务。两个使用 TCP 软件的服务之间的数据传输单位称为段。TCP 连接包含三个阶段：连接建立、数据传输和连接终止。TCP 软件通常以有限状态机（FSM）形式实现。

3.6　习题集

测试题

本章的交互测试题集可以在本书的网站上找到。强烈推荐学生们做这些测试题，这样可以在学生做习题集前检测他们对课程内容的理解。

练习题

Q3-1　假设在一个系统中我们有一组专用计算机，每台都用来执行单独一个任务。我们仍然需要主机到主机和进程到进程的通信以及两级地址吗？

Q3-2　操作系统给每个正在运行的应用程序分配一个进程号。你能解释为什么这些进程号不能代替端口号吗？

Q3-3　假设你需要在你家的两台主机上编写并测试一个客户-服务器应用程序。

a．你将为客户程序选择什么范围的端口号？

b．你将为主机程序选择什么范围的端口号？

c．两个端口号可以相同吗？

Q3-4　假设一个新的组织需要创建一个新的服务器进程并允许它的客户使用这个进程访问组织站点。服务器进程的端口号应该如何选择？

Q3-5　在网络中，接收窗口的大小是 1 个分组。以下哪个协议可以被网络使用？

a．Stop-and-Wait　　**b**．Go-Back-*N*　　**c**．Selective-Repeat

Q3-6　在网络中，发送窗口的大小是 20 个分组。以下哪个协议可以被网络使用？

a．Stop-and-Wait　　**b**．Go-Back-*N*　　**c**．Selective-Repeat

Q3-7　在 m 是定值且 $m > 1$ 的网络中，我们可以使用回退 *N* 帧或选择性重复协议。请描述这两者的优缺点。在选择这些协议的时候还需要考虑什么其他网络标准？

Q3-8　既然存储分组序号的字段在大小上是有限的，那么协议中的序号就需要回绕，这意味着两个分组可能有相同的序号。在一个序号字段是 m 位的协议中，如果分组具有序号 x，需要发送多少的分组才能看到一个相同的序号 x，假设每个分组都被分配一个序号。

Q3-9　我们在前一个问题中讨论的回绕情况造成了一个网络中的问题吗？

Q3-10　你能解释为什么某些传输层分组可能在因特网中失序接收吗？

Q3-11　你能解释为什么某些传输层分组可能在因特网中丢失吗？

Q3-12　你能解释为什么某些传输层分组可能在因特网中重复吗？

Q3-13　在回退 *N* 帧协议中，发送窗口的大小可能是 $2^m - 1$，而接收窗口大小只是 1。当发送和接收窗口大小有很大差别时，流量控制是如何实现的？

Q3-14　在选择性重复协议中，发送和接收窗口的大小是相同的。这是否意味着应该没有分组处于运输中呢？

Q3-15　一些应用程序可以使用两种传输层协议（UDP 或 TCP）。当一个分组到达目的端，计算机如何找出它

使用哪种传输层协议？

Q3-16 一个位于 IP 地址为 122.45.12.7 主机的客户发送一个报文到相应服务器，服务器位于 IP 地址为 200.112.45.90 的主机。如果熟知端口号是 161 且临时端口是 51000，通信中使用的套接字地址对是什么？

Q3-17 UDP 是一个面向报文的协议。TCP 是一个面向字节的协议。如果一个应用需要保护报文边界，应该使用什么协议，UDP 还是 TCP？

Q3-18 在第 2 章，我们说我们可以使用不同的应用程序接口（API），如套接字接口、TLI 和 STREAM，这些可以用来提供客户–服务器通信。这是不是意味着我们应该使用支持这些 API 的不同的 UDP 或 TCP 协议？请解释。

Q3-19 假设一个私有网络，主机之间使用点到点通信并且不需要路由，它完全不使用网络层。这个网络仍然能从 UDP 或 TCP 中获益吗？换言之，用户数据报或段能被封装到以太帧中吗？

Q3-20 假设一个私有网络使用与 TCP/IP 完全不同的协议簇。这个网络也能使用 UDP 或 TCP 服务作为端到端报文通信的媒介吗？

Q3-21 你能解释在 TCP 中为什么需要 4（有时是 3 个）段来结束连接吗？

Q3-22 在 TCP 中，一些段类型只能被用来控制；它们不能同时携带数据。你能定义一些这样的段吗？

Q3-23 在 TCP 中，我们如何定义段的序号（在每个方向上）？考虑两种情况：第一个段和其他段。

Q3-24 在 TCP 中，我们有两个连续段。假设第一个段的序号是 101。以下每种情况中，下一个段的序号是多少？

a．第一个段不消耗序号。

b．第一个段消耗 10 个序号。

Q3-25 在 TCP 中以下每个段消耗多少序号？

a．SYN　　**b**．ACK　　**c**．SYN + ACK　　**d**．Data

Q3-26 你能解释在 TCP 中为什么 SYN、SYN + ACK 以及 FIN 段，每个段消耗一个序号，但是一个不携带数据的 ACK 段不消耗序号吗？

Q3-27 观察 TCP 头部（见图 3-44），当窗口大小只有 16 位长时，我们发现序号是 32 位长。这是不是意味着 TCP 在这方面更接近回退 N 帧或选择性重复协议？

Q3-28 TCP 中的窗口最大值起初被设计为 64KB（这意味着 $64 \times 1024 = 65\,536$ 或者实际上为 65 535）。你能说出这里的原因吗？

Q3-29 TCP 头部大小的最大值是多少？　TCP 头部大小的最小值是多少？

Q3-30 在 TCP 中，SYN 段是在一个方向上打开连接还是两个方向？

Q3-31 在 TCP 中，FIN 段是在一个方向上关闭连接还是两个方向？

Q3-32 在 TCP 中，什么类型的标志位可以完全关闭双向的通信？

Q3-33 绝大多数的标志位可以在段中一起使用。给出一个由于歧义不能同时使用两种标志位的例子。

Q3-34 假设一个客户向服务器发送一个 SYN 段。当服务器检查数值端口号时，它发现没有端口号所定义的正在的运行进程。这种情况下，服务器应该怎么做？

Q3-35 你能解释为什么 TCP 使用了不可靠 IP 提供的服务，却能够提供可靠通信吗？

Q3-36 我们说 TCP 提供了两个应用程序之间的面向连接服务。在这种情况下，连接需要连接标识符用以相互区分。你认为这种情况下独立的连接标识符是什么？

Q3-37 假设 Alice 使用她的浏览器打开了两个通向 Bob 服务器上运行着的 HTTP 服务器的连接。TCP 如何区分这两个连接？

Q3-38 在讨论使用 TCP 的面向连接通信中，我们使用被动打开和主动打开这两个术语。假设 Alice 和 Bob 之间有电话交谈，由于电话交谈是面向连接通信的一个例子，因此假设 Alice 打电话给 Bob 并且他们在电话上交谈。在这种情况下谁在进行一个被动打开连接？谁在进行一个主动打开连接？

Q3-39 在 TCP 中，发送窗口可以小于、大于或等接收窗口吗？

Q3-40 你能举出一些由一个 TCP 段或一些 TCP 段组合完成的任务吗？

Q3-41 在 TCP 段中，序号表示什么？

Q3-42 在 TCP 段中，确认号表示什么？

Q3-43 差错控制中使用的校验和在以下协议中是可选的还是强制的？

a．UDP　　**b**．TCP

Q3-44　假设一个 TCP 服务器期待接收 2001 字节，但是它接收了序号为 2200 的段。TCP 服务器对这个事件的反应是什么？你能证明这个反应吗？

Q3-45　假设一个 TCP 客户期待接收 2001 字节，但是它接收了序号为 1201 的段。TCP 客户对这个事件的反应是什么？你能证明这个反应吗？

Q3-46　假设一个 TCP 服务器期待接收 2001 到 3000 字节，但是它接收了序号为 2001 携带 400 字节的段。TCP 服务器对这个事件的反应是什么？你能证明这个反应吗？

Q3-47　假设一个 TCP 服务器期待接收 2401 字节，但是它接收了序号为 2401 携带 500 字节的段。如果此时服务器没有数据要发送且没有确认之前的段，TCP 服务器对这个事件的反应是什么？你能证明这个反应吗？

Q3-48　假设一个 TCP 客户期待接收 3001 字节，但是它接收了序号为 3001 携带 400 字节的段。如果此时客户没有数据要发送且没有确认之前的段，TCP 客户对这个事件的反应是什么？你能证明这个反应吗？

Q3-49　假设一个 TCP 服务器期待接收 6001 字节，但是它接收了序号为 6001 携带 2000 字节的段。如果此时服务器要发送 4001 到 5000 字节。TCP 服务器对这个事件的反应是什么？你能证明这个反应吗？

Q3-50　为 TCP 产生 ACK 的第一条规则没有在图 3-59 或图 3-60 中给出。请解释原因。

Q3-51　在我们表述的产生 ACK 的六条规则中，哪一条可以应用于服务器从客户接收到 SYN 段？

Q3-52　在我们表述的产生 ACK 的六条规则中，哪一条可以应用于客户从服务器接收到 SYN + ACK 段？

Q3-53　在我们表述的产生 ACK 的六条规则中，哪一条可以应用于客户或服务器从另一端接收到 FIN 段？

思考题

P3-1　比较 16 位地址的范围（即 0 到 65 535）与 32 位 IP 地址的范围（即 0 到 4 294 967 295）（第 4 章讨论）。为什么我们需要这么大范围的 IP 地址，但只需要相对较小的端口号范围？

P3-2　你能解释为什么 ICANN 将端口号分成三组：熟知、注册和动态吗？

P3-3　发送端向同一个目的发送一系列分组，使用 5 位序号。如果序号从 0 开始，第 100 个分组的序号是多少？

P3-4　在如下的每个协议中，在回绕发生前可以有多少分组拥有独立的序号？

a．Stop-and-Wait　　**b**．Go-Back-*N* 其中 $m = 8$　　**c**．Select-Repeat 其中 $m = 8$

P3-5　使用 5 位序号，以下协议中发送和接收窗口最大是多少？

a．Stop-and-Wait　　**b**．Go-Back-*N*　　**c**．Select-Repeat

P3-6　给出一个带有三种状态的假想状态机的有限状态机：A（开始状态）、B 和 C；以及四个事件：事件 1、2、3 和 4。以下指定了状态机的行为：

a．当处于状态 A 时，可能发生两个事件：事件 1 和事件 2。如果事件 1 发生，状态机执行动作 1 并进入状态 B。如果事件 2 发生，状态机进入状态 C（无动作）。

b．当处于状态 B 时，可能发生两个事件：事件 3 和事件 4。如果事件 3 发生，状态机执行动作 2 并进入状态 B。如果事件 4 发生，状态机进入状态 C。

c．当处于状态 C 时，状态机永远保持这种状态。

P3-7　假设我们的网络永远不会出现被破坏、丢失或重复分组。我们只考虑流量控制。我们不想让发送方用分组淹没接收方。设计一个有限状态机，当接收方准备好时允许发送方给接收方发送一个分组。如果接收方准备好接收一个分组，它就发送一个 ACK。没有接收到发给发送方的 ACK 就表示接收方没有准备好接收更多的分组。

P3-8　假设我们的网络永远不会出现被破坏、丢失或重复分组。我们也只考虑流量控制。我们不想让发送方用分组淹没接收方。设计一个新协议的有限状态机来实现这些特性。

P3-9　在停止–等待协议中，给出接收方接收到一个重复分组（也是失序的）的情况。提示：考虑延迟 ACK。接收方对这个事件的反应是什么？

P3-10　假设我们想改变停止–等待协议并加入 NAK（消极 ACK）分组到系统中。当一个被破坏分组到达接收方，接收方丢弃这个分组，但是发送一个 NAK，其中 nakNo 定义了被破坏分组的 seqNo。按这种方式，发送方能够重发被破坏分组而不用等待超时。请解释图 3-21 中的有限状态机需要作何改变且用时间线图给出实现这个新协议的例子。

P3-11　重画图 3-19，其中 5 个分组被交换（0、1、2、3 和 4）。假设分组 2 丢失且分组 3 在分组 4 之后到达。

P3-12　创建一个与图 3-22 相似的场景，其中发送方发送三个分组。第一个和第二个分组到达且被确认。第

三个分组延迟并被重发。在对原始分组的确认被发送后，重复分组被接收到。

P3-13 创建一个与图 3-22 相似的场景，其中发送方发送两个分组。第一个分组被接收到且被确认，但是确认丢失。发送方在超时后重发分组。第二个分组丢失并且重发。

P3-14 重画图 3-29，其中发送方发送 5 个分组（0、1、2、3 和 4）。分组 0、1 和 2 被发送且被一个 ACK 确认，这个 ACK 在所有分组被发送后到达发送端。分组 3 被接收且被一个 ACK 确认。分组 4 丢失并被重发。

P3-15 重画图 3-35，其中发送方发送 5 个分组（0、1、2、3 和 4）。分组 0、1 和 2 被按序发送且被一个一个地确认。分组 3 被延迟并在分组 4 之后被接收。

P3-16 回答如下与停止–等待协议有限状态机相关的问题（见图 3-21）：

a．发送状态机处于准备状态且 $S = 0$。下一个待发送分组的序号是多少？

b．发送状态机处于阻塞状态且 $S = 1$。如果发生超时下一个待发送分组的序号是多少？

c．接收状态机处于准备状态且 $R = 1$。序号为 1 的分组到达。回应这个事件的动作是什么？

d．接收状态机处于准备状态且 $R = 1$。序号为 0 的分组到达。回应这个事件的动作是什么？

P3-17 回答如下与回退 N 帧协议 $m = 6$ 位的有限状态机相关的问题。假设窗口大小是 63（见图 3-27）：

a．发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。下一个待发送分组的序号是多少？

b．发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。超时发生。有多少分组待重发？它们的序号是多少？

c．发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。一个 ackNo = 13 的 ACK 到达。下一个 S_f 和 S_n 的值是多少？

d．发送状态机处于阻塞状态且 $S_f = 14$，$S_n = 21$。窗口大小是多少？

e．发送状态机处于阻塞状态且 $S_f = 14$，$S_n = 21$。一个 ackNo = 18 的 ACK 到达。下一个 S_f 和 S_n 的值是多少？发送状态机的状态是什么？

f．接收方状态机处于准备状态且 $R_n = 16$。序号为 16 的分组到达。下一个 R_n 的值是多少？状态机对这个事件的响应是什么？

P3-18 回答如下与回退 N 帧协议 $m = 7$ 位的有限状态机相关的问题。假设窗口大小是 64（图 3-34）：

a．发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。下一个待发送分组的序号是多少？

b．发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。分组 10 的计时器超时。有多少分组待重发？它们的序号是多少？

c．发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。一个 ackNo = 13 的 ACK 到达。下一个 S_f 和 S_n 的值是多少？回应这个事件的动作是什么？

d．发送状态机处于阻塞状态且 $S_f = 14$，$S_n = 21$。窗口大小是多少？

e．发送状态机处于阻塞状态且 $S_f = 14$，$S_n = 21$。一个 ackNo = 18 的 ACK 到达。分组 15 和 16 已经被确认，下一个 S_f 和 S_n 的值是多少？发送状态机的状态是什么？

f．接收方状态机处于准备状态且 $R_n = 16$。窗口大小为 8。序号为 16 的分组到达。下一个 R_n 的值是多少？状态机对这个事件的响应是什么？

P3-19 我们将网络中的带宽延迟乘积定义为在往返时间（RTT）内可以处于管道中的分组数。以下每种情况中的带宽延迟乘积是多少？

a．带宽：1 Mbps，RTT：20 ms，分组大小：1000 位

b．带宽：10 Mbps，RTT：20 ms，分组大小：2000 位

c．带宽：1 Gbps，RTT：4 ms，分组大小：10 000 位

P3-20 假设我们需要为一个网络设计回退 N 帧滑动窗口协议，这个网络带宽是 100Mbps 且发送方和接收方之间的平均距离是 10 000km。假设平均分组大小是 100 000 位且介质中的传输速度是 2×10^8m/s。找出发送和接收窗口的最大值，序号字段的位数（m）以及计时器适当的超时值。

P3-21 假设我们需要为一个网络设计选择性重复滑动窗口协议，这个网络带宽是 1Gbps 且发送方和接收方之间的平均距离是 5 000km。假设平均分组大小是 50 000 位且介质中的传输速度是 2×10^8m/s。找出发送和接收窗口的最大值，序号字段的位数（m）以及计时器的适当超时值。

P3-22 回退 N 帧协议中的确认号定义了下一个预期分组，但是选择性重复协议中的确认号定义了被确认的序号。你能解释原因吗？

P3-23　在使用回退 N 帧协议的网络中 $m = 3$ 且发送窗口的大小是 7，变量值为 $S_f = 62$，$S_n = 66$ 且 $R_n = 64$。假设网络不复制或重排分组，

a．正在传输的数据分组的序号是多少？　　**b**．正在传输的 ACK 分组确认号是多少？

P3-24　在使用选择性重复协议的网络中 $m = 4$ 且发送窗口的大小是 8，变量值为 $S_f = 62$，$S_n = 67$ 且 $R_n = 64$。分组 65 已经在发送端被确认；分组 65 和 66 被接收端失序接收。假设网络没有复制分组。

a．即将来临的数据分组的序号是多少（正在传输、被损坏或丢失）？

b．即将来临的 ACK 分组的确认号是多少（正在传输、被损坏或丢失）？

P3-25　回答下列问题：

a．UDP 用户数据报的最小是大？

b．UDP 用户数据报的最大是大？

c．可以被封装到 UDP 用户数据报中的应用层有效载荷数据最小是多大？

d．可以被封装到 UDP 用户数据报中的应用层有效载荷数据大是多大？

P3-26　一个客户使用 UDP 向服务器发送数据。数据长度是 16 字节。计算 UDP 层的传输效率（有用字节与总字节的比）。

P3-27　以下是 UDP 头部十六进制格式的转储（内容）。

0045DF0000580000

a．源端口号是多少？

b．目的端口号是多少？

c．用户数据报的总长度是多少？

d．数据长度是多少？

e．分组是从客户到服务器的吗？还是相反的？

f．应用层协议是什么？

g．发送方计算这个分组的校验和了吗？

P3-28　比较 TCP 和 UDP 头部。列出 TCP 头部中 UDP 头部所没有的字段。给出 UDP 头部中没有这些字段的原因。

P3-29　在 TCP 中，如果 HLEN 的值为 0111，被包含进段的选项有多少字节？

P3-30　以下 TCP 段是什么类型的？其中控制字段的值为：

a．000000　　**b**．000001　　**c**．010001

d．000100　　**e**．000010　　**f**．010010

P3-31　TCP 段中的控制字段是 6 位。我们可以有 64 中不同的组合。列出你认为通常使用的组合。

P3-32　以下是 TCP 头部十六进制格式的转储（内容）。

E293 0017 00000001 00000000 5002 07FF…

a．源端口号是多少？　　**b**．目的端口号是多少？

c．序号是多少？　　**d**．确认号是多少？

e．头部长度是多少？　　**f**．段类型是什么？

g．窗口大小是多少？

P3-33　为了更好地理解三次握手连接建立的必要性，让我们仔细检查场景。Alice 和 Bob 不能打电话或上网（想一下过去的日子）来建立他们下一次会见，这个会见是在离家很远的一个地方。

a. 假设 Alice 发送了一封信给 Bob 并确定了他们会见的日期和时间。Alice 能够到会见地点并确定 Bob 也在吗？

b. 假设 Bob 用信件回复了 Alice 的请求并确认日期和时间。Bob 能够到会见地点并确定 Alice 也在吗？

c. 假设 Alice 回复了 Bob 的请求并确认相同的日期和时间。他们中的一个人能够到会见地点并确定对方也在吗？

P3-34　为了使初始序号是随机数字，绝大多数系统在启动阶段从 1 开启一个计数器并每半秒增加 64 000。计数器回绕一次需要多长时间？

P3-35　在 TCP 连接中，客户端的初始序号是 2171。客户打开连接，发送三个段，第二个段携带 1000 字节数据，并关闭连接。以下客户发出的每个段的序号是多少？

a．SYN 段　　b．数据段　　c．FIN 段

P3-36 在连接中，cwnd 的值为 3000 且 rwnd 为 5000。主机已经发送 2000 字节，没有被确认。可以再发送多少字节？

P3-37 一个客户使用 TCP 发送数据给服务器。数据包含 16 字节。计算 TCP 层的传输效率（有用字节与总字节的比）。

P3-38 TCP 正在以每秒 1 兆字节速度发送数据。如果序号从 7000 开始，序号回到 0 需要多长时间？

P3-39 一个 HTTP 客户使用初始序号（ISN）14 534 打开一个 TCP 连接，并且临时端口号是 59 100。服务器打开连接，它的 ISN 为 21 732。如果 rwnd 是 4000 且服务器定义 rwnd 为 5000，请给出连接建立期间的三个 TCP 段。忽略校验和字段的计算。

P3-40 假设在之前问题中的 HTTP 客户发送一个 100 字节的请求。服务器回应了一个 1200 字节的段。给出客户和发送方之间交换的两个段的内容。假设响应的确认将在稍后被客户完成（忽略校验和字段的计算）。

P3-41 假设在之前问题中的 HTTP 客户关闭连接，并且同时对从服务器收到的响应中的字节进行确认。在接收到客户端的 FIN 段之后，服务器也关闭另一个方向的连接。给出连接终止阶段。

P3-42 区分超时事件和三次重复 ACK 事件。哪一个是网络中拥塞较严重的标志？为什么？

P3-43 图 3-52 给出了状态转换图中的客户和服务器，这幅状态转换图描述的是使用四次握手关闭的普通场景。改变这幅图来给出三次握手关闭。

P3-44 Eve，这位入侵者，她使用 Alice 的 IP 地址，向 Bob 这台服务器发送一个 SYN 段。Eve 能够通过假装她是 Alice 来与 Bob 建立 TCP 连接吗？假设 Bob 为每次连接使用一个不同的 ISN。

P3-45 Eve，这位入侵者，她使用 Alice 的 IP 地址，给 Bob 这台服务器发送一个用户数据报。Eve 能够假装她是 Alice 来接收来自 Bob 的响应吗？

P3-46 假设 Alice，这个客户，与 Bob 这台服务器创建了一个连接。他们交换数据并关闭连接。现在 Alice 通过发送一个新的 SYN 段开始一个与 Bob 的新连接，在 Bob 回复这个 SYN 段之前，一个来自 Alice 的旧 SYN 段的副本到达了 Bob 的计算机，这个段之前在网络中慢慢前行，它发起了来自 Bob 的 SYN + ACK。这个段会被 Alice 的计算机误认为是对新 SYN 段的响应吗？请解释。

P3-47 假设 Alice，这个客户，与 Bob 这台服务器建立了 TCP 连接。他们交换数据并关闭连接。现在 Alice 通过发送一个新的 SYN 段开始一个与 Bob 的新连接。服务器响应 SYN + ACK 段。然而，在 Bob 接收来自 Alice 段对这个连接的 ACK 之后，一个重复的旧 ACK 段从 Alice 到达 Bob 端。这个旧的 ACK 会与 Bob 期待从 Alice 端接收的 ACK 混淆吗？

P3-48 使用图 3-56，请解释在 TCP 的发送端如何实现流量控制（从发送 TCP 到发送应用程序）。请画图。

P3-49 使用图 3-56，请解释在 TCP 的接收端如何实现流量控制（从发送 TCP 到发送应用程序）。

P3-50 在 TCP 中，假设一个客户有 100 字节要发送。客户每 10ms 创建 10 字节并将它们传递到应用层。服务器立即确认每个段或计时器在 50ms 超时服务器确认每个段。如果实现使用最大段长度（MSS）是 30 字节的 Nagle 算法，给出段和每个段携带的字节。往返时间是 20ms，但是发送方计时器被设置成 100ms。有携带最大长度的段吗？Nagle 算法真的有效吗？为什么？

P3-51 为了更好地看清 Nagle 算法，让我们重复之前的问题，但是当之前的段（每隔一个段）没有被确认或计时器在 60ms 后超时，我们让服务器传输层确认段。给出这个场景的时间线。

P3-52 正如我们在文中解释的，当不使用新 SACK 选项时，TCP 滑动窗口是回退 *N* 帧和选择性重复协议的组合。请解释 TCP 滑动窗口中哪个方面与回退 *N* 帧接近，哪个方面与选择性重复协议接近？

P3-53 尽管新 TCP 实现使用 SACK 选项来报告字节的失序和重复范围，但是请解释旧实现如何表示接收段中失序的或重复的字节。

P3-54 我们在本书网站讨论 TCP 的 SACK 新选项，但是暂时假设我们添加了一个 8 字节 NAK 选项到 TCP 段的尾部，TCP 段可以支持 32 位序号。请给出我们如何使用这个 8 字节 NAK 来定义接收到的字节中失序或重复范围。

P3-55 在 TCP 连接中，假设最大段长度（MSS）是 1000 字节。客户进程有 5400 字节要发送到服务器进程，没有字节要响应（单向通信）。TCP 服务器根据我们在文中讨论的规则产生 ACK。给出慢启动阶段事务的时间线，给出开始、结束以及每次变化后的 cwnd 值。假设每个段头部只有 20 字节。

P3-56 Taho TCP 站的 ssthresh 值被设置为 6MSS。现在站处于慢启动状态，cwnd = 4MSS。给出在如下事件

之前和之后的 cwnd、ssthresh 的数值以及站状态。这些事件是：四个连续非重复 ACK 到达之后是一个超时；四个连续非重复 ACK 到达之后是三次非重复 ACK。

P3-57　Reno TCP 站的 ssthresh 值被设置为 8MSS。现在站处于慢启动状态，cwnd = 5MSS 且 ssthresh = 8 MSS。给出在如下事件之后的 cwnd、ssthresh 的数值以及站的当前和下一个状态。这些事件是：三个连续非重复 ACK 到达之后是五个重复 ACK；三个连续非重复 ACK 到达之后是两个非重复 ACK；三个连续非重复 ACK 到达之后是一个超时。

P3-58　在 TCP 连接中，窗口大小在 60 000 字节和 30 000 字节之间波动。如果平均 RTT 是 30ms，连接的吞吐量是多少？

P3-59　如果初始 RTT_S = 14ms 且α被设置为 0.2，在如下事件之后计算新 RTT_S（时间与事件 1 相关）：

事件 1：00 ms　　段 1 被发送。
事件 2：06 ms　　段 2 被发送。
事件 3：16 ms　　段 1 超时且被重发。
事件 4：21 ms　　段 1 被确认。
事件 5：23 ms　　段 2 被确认。

P3-60　在连接中，假设旧 RTT_D = 7ms。如果新 RTT_S = 17 且新 RTT_M = 20，新的 RTT_D 是多少？令β= 0.25。

3.7　模拟实验

Applets

我们构建了一些 Java 小程序用于展示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些实际的协议。

实验作业

在本章，我们使用 Wireshark 来捕捉并调查一些传输层分组。我们使用 Wireshark 来模拟两个协议：UDP 和 TCP。

Lab3-1　在第 1 个实验中，我们使用 Wireshark 来捕捉运行中的 UDP 分组。我们检查每个字段的值并检查是否校验和被计算。

Lab3-2　在第 2 个实验中，我们使用 Wireshark 来捕捉并仔细研究 TCP 分组中的很多特性。这些特性包括可靠性、拥塞控制以及流量控制。Wireshak 让我们看到 TCP 如何使用段中的序号和确认号来实现可靠数据传输。我们也能观察运行中的 TCP 拥塞算法（慢启动、拥塞避免以及快速恢复）。另一个 TCP 的特性是流量控制。我们观察 TCP 中沿着从接收方到发送方方向的流量控制是通过接收方通告的 cwnd 数值实现的。

3.8　编程作业

利用你选择的编程语言，编写源代码，编译并测试如下程序：

Prg3-1　编写程序来模拟位于发送端的简单协议的有限状态机（见图 3-18）。
Prg3-2　编写程序来模拟位于发送端的停止–等待协议的有限状态机（见图 3-21）。
Prg3-3　编写程序来模拟位于发送端的回退 *N* 帧协议的有限状态机（见图 3-27）。
Prg3-4　编写程序来模拟位于发送端的选择性重复协议的有限状态机（见图 3-34）。

计算机网络教程 自顶向下方法

Computer Networks A Top-Down Approach

本书是计算机领域知名作者Forouzan按照目前计算机网络教学比较流行的自顶向下方法编写的一部重要教材，它延续了作者一贯的写作风格，以通俗易懂的方式全面阐述了计算机网络原理及其应用，并介绍了一些目前计算机网络发展的新技术。此外，每章都配有丰富的习题集（包括测试题、练习题、思考题），部分章节还包含模拟实验和编程作业，有助于读者巩固所学知识，提高动手实践能力。

本书特色

- 协议分层：本书利用Internet协议分层和TCP/IP协议簇讲授网络原理，强调各层网络理论之间的相互关系。
- 自顶向下：从应用层开始，尽早让读者理解网络设备如何工作，然后讨论其他各层，最后介绍物理层。
- 形象直观：采用图文并茂的方法描述技术性很强的问题，较少涉及复杂的数学公式，便于读者理解相关概念。
- 举例和应用：以丰富的实例阐明相关概念，并添加了一些现实中的应用，激发读者的学习热情。

作者简介

Behrouz A. Forouzan　毕业于加州大学艾尔温分校，现在是迪安那大学教授，从事计算机信息系统专业的课程设置。此外，他还是多家公司的系统开发咨询顾问。除本书外，Forouzan还著有多部成功的计算机科学方面的书，包括《Data Communications and Networking》、《Foundations of Computer Science》、《TCP/IP Protocol Suite》等。



影印版
书号：978-7-111-37430-5
定价：79.00元

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

Mc Graw Hill Education
www.mheducation.com

上架指导：计算机科学/网络
ISBN 978-7-111-40088-2
9 787111 400882
定价：99.00元